

Probabilistic Programming and its Applications

Luc De Raedt

with many slides from Angelika Kimmig

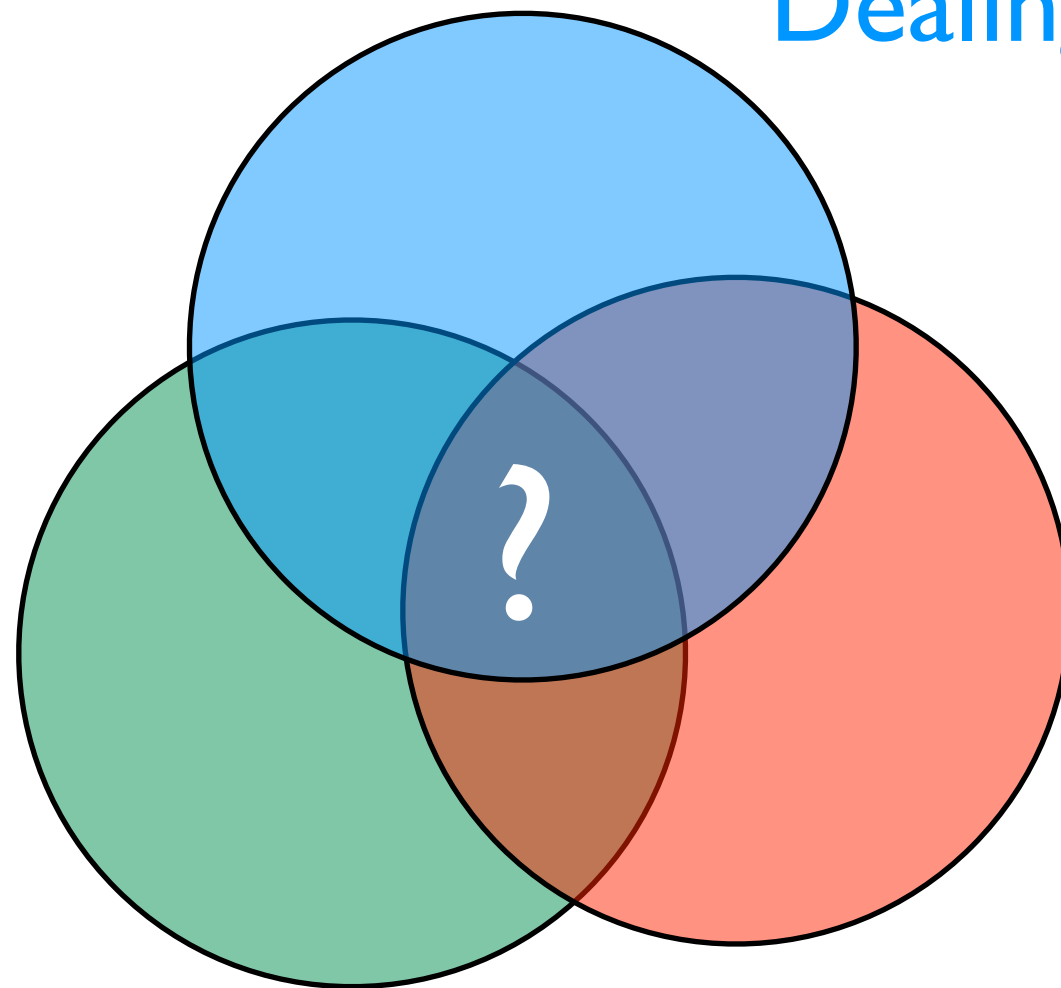


KU LEUVEN

A key question in AI:

Dealing with uncertainty

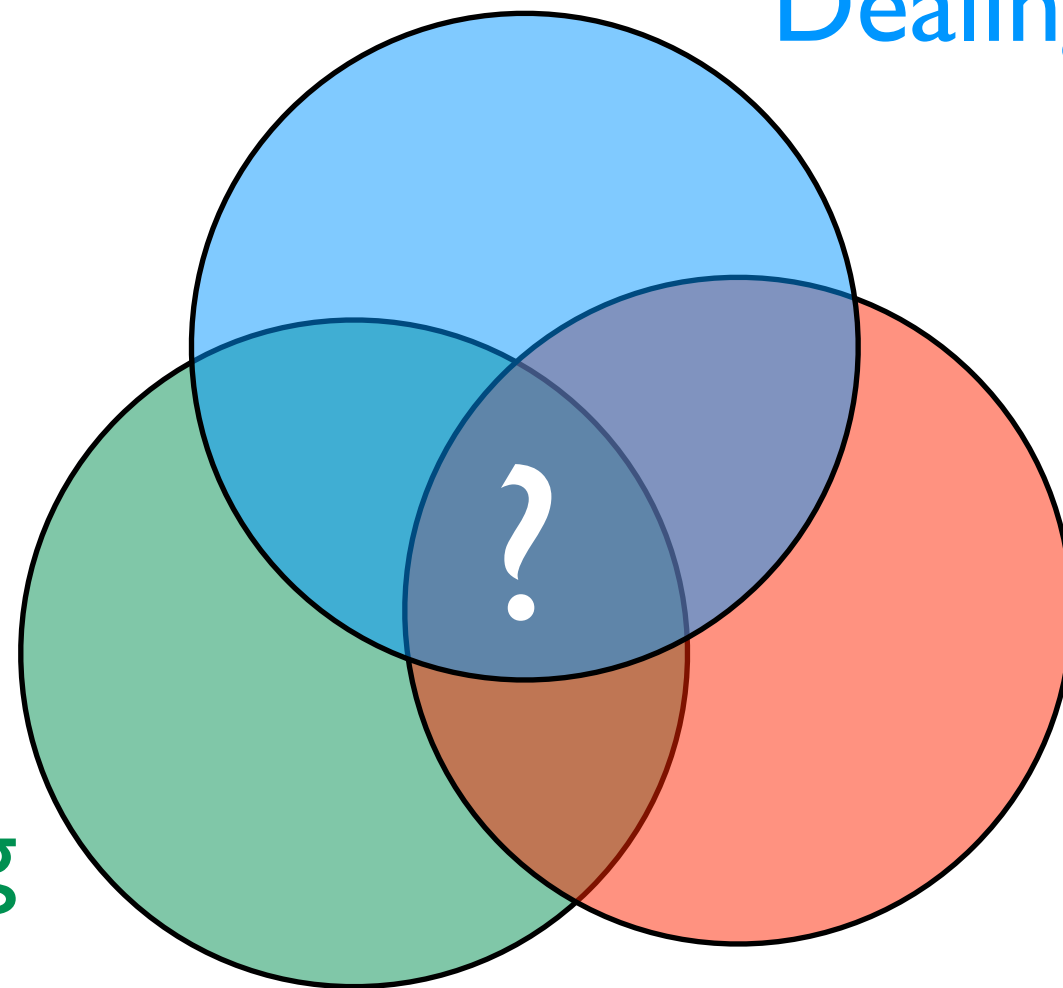
Reasoning with
relational data



Learning

A key question in AI:

Dealing with uncertainty



Learning

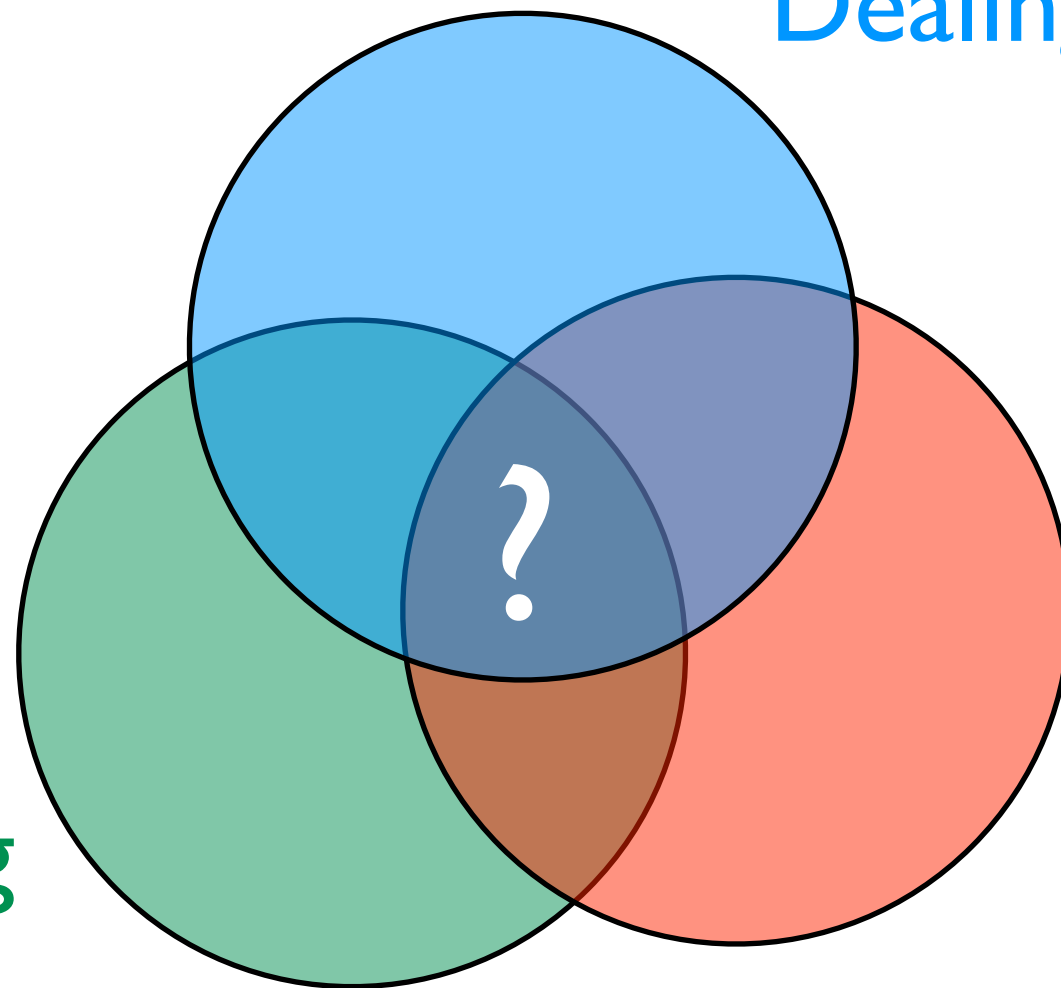
Reasoning with
relational data

- logic
- databases
- programming
- ...

A key question in AI:

Reasoning with
relational data

- logic
- databases
- programming
- ...



Dealing with uncertainty

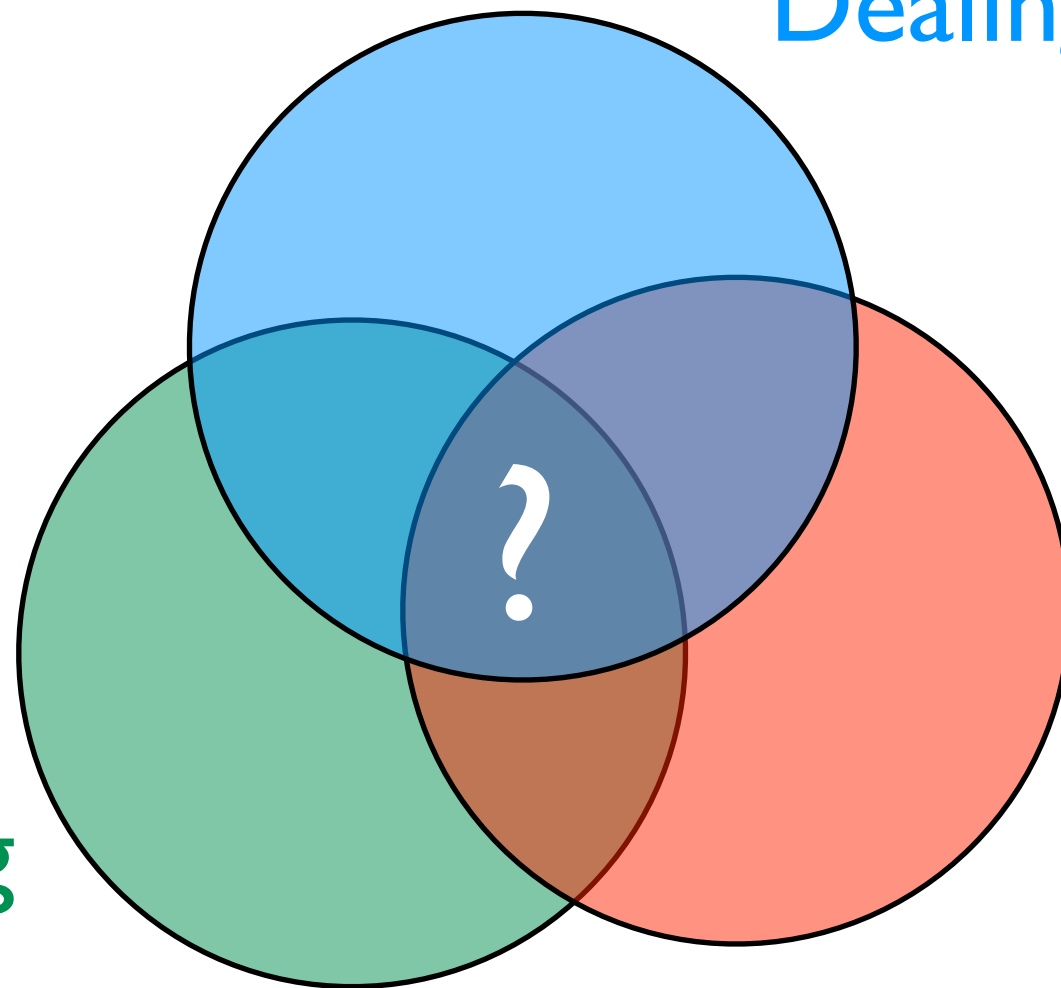
- probability theory
- graphical models
- ...

Learning

A key question in AI:

Reasoning with relational data

- logic
- databases
- programming
- ...



Dealing with uncertainty

- probability theory
- graphical models
- ...

Learning

- parameters
- structure

A key question in AI:



Statistical relational learning
& Probabilistic Programming

The need for relations

Dynamics: Evolving Networks



- *Travian*: A massively multiplayer real-time strategy game
 - Commercial game run by TravianGames GmbH
 - ~3.000.000 players spread over different “worlds”
 - ~25.000 players in one world

[Thon et al. ECML 08]



World Dynamics

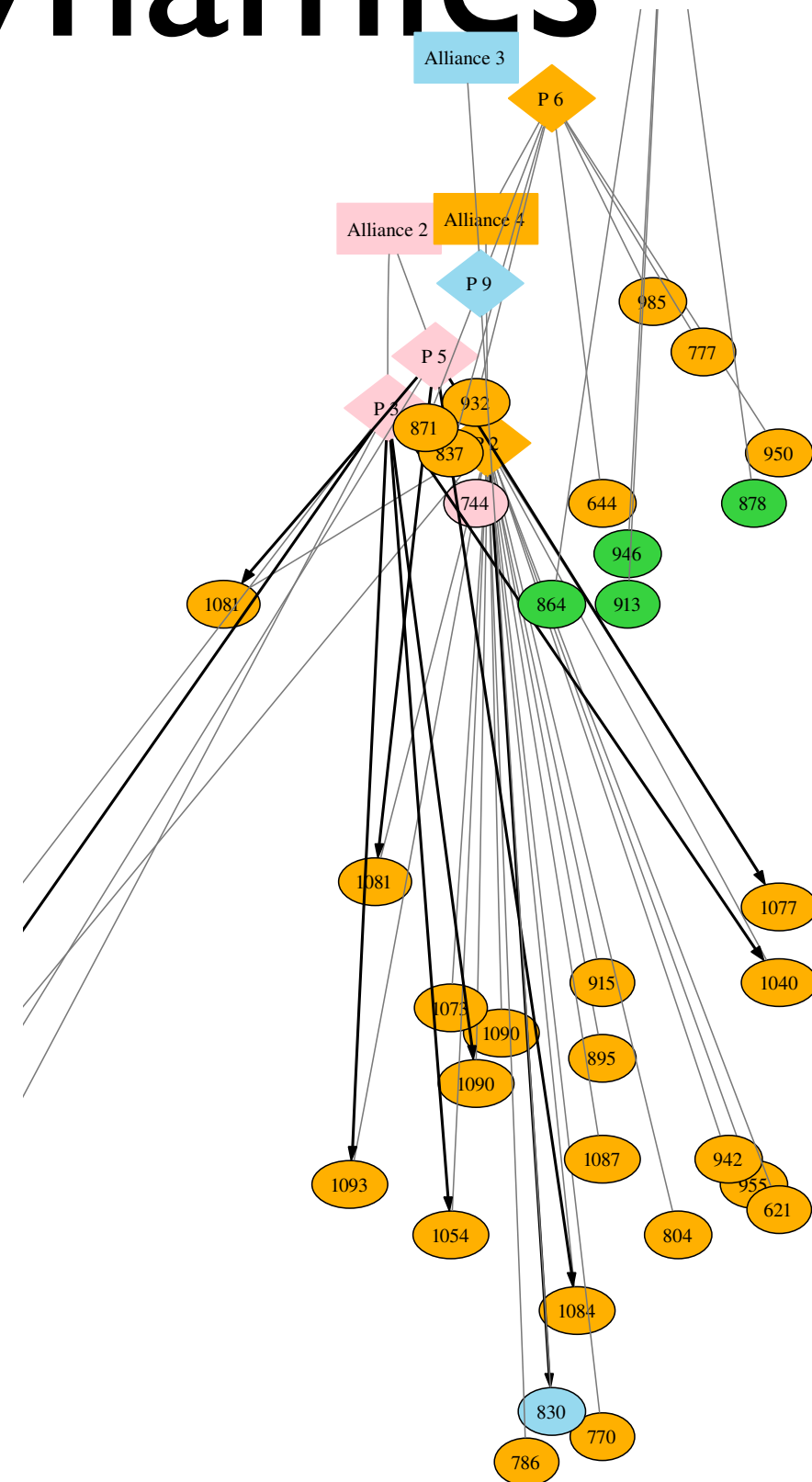
Fragment of world with

- ~10 alliances
- ~200 players
- ~600 cities

alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?

[Thon, Landwehr, De Raedt, ECML08]



World Dynamics

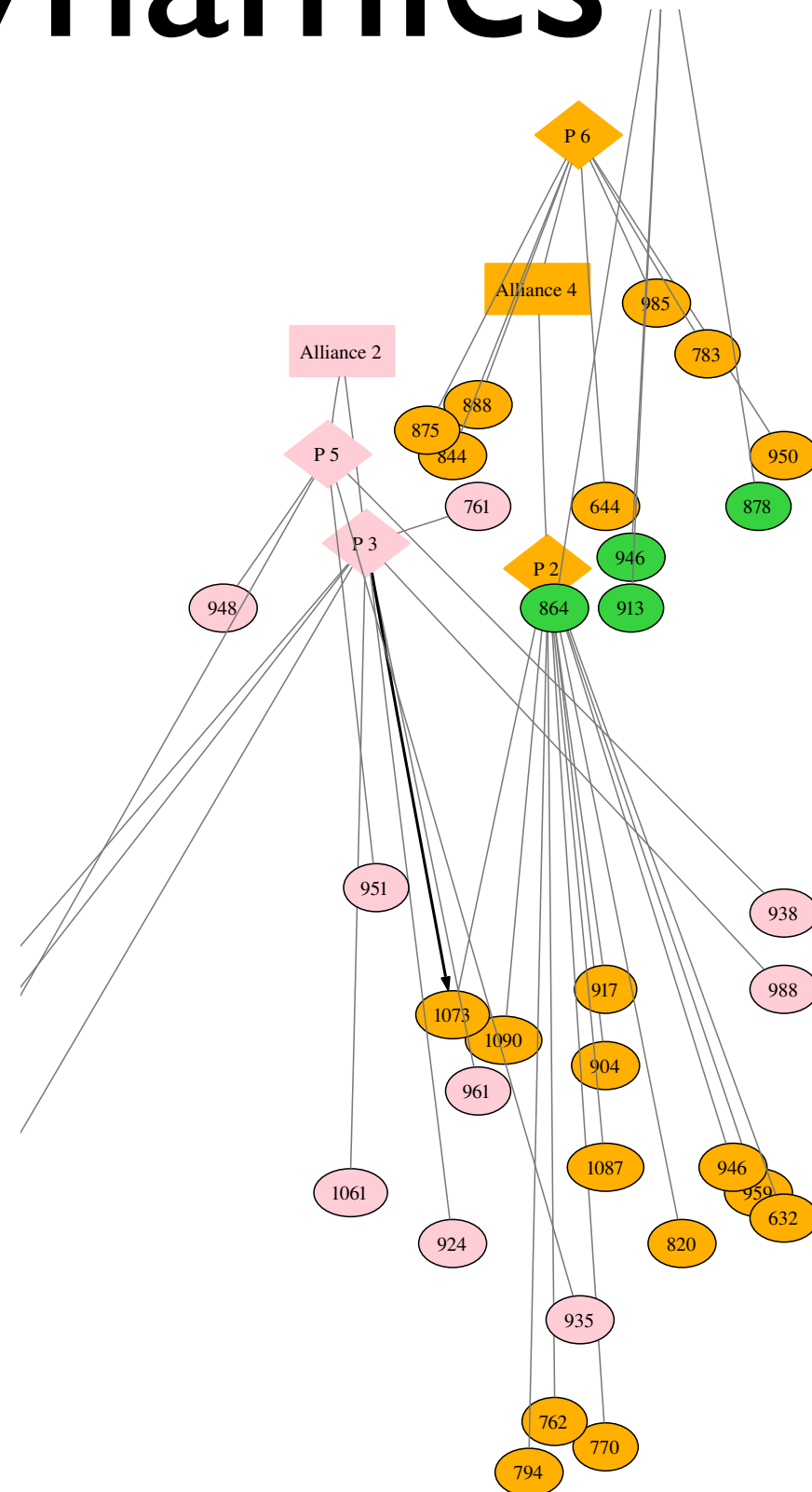
Fragment of world with

- ~10 alliances
- ~200 players
- ~600 cities

alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?

[Thon, Landwehr, De Raedt, ECML08]



World Dynamics

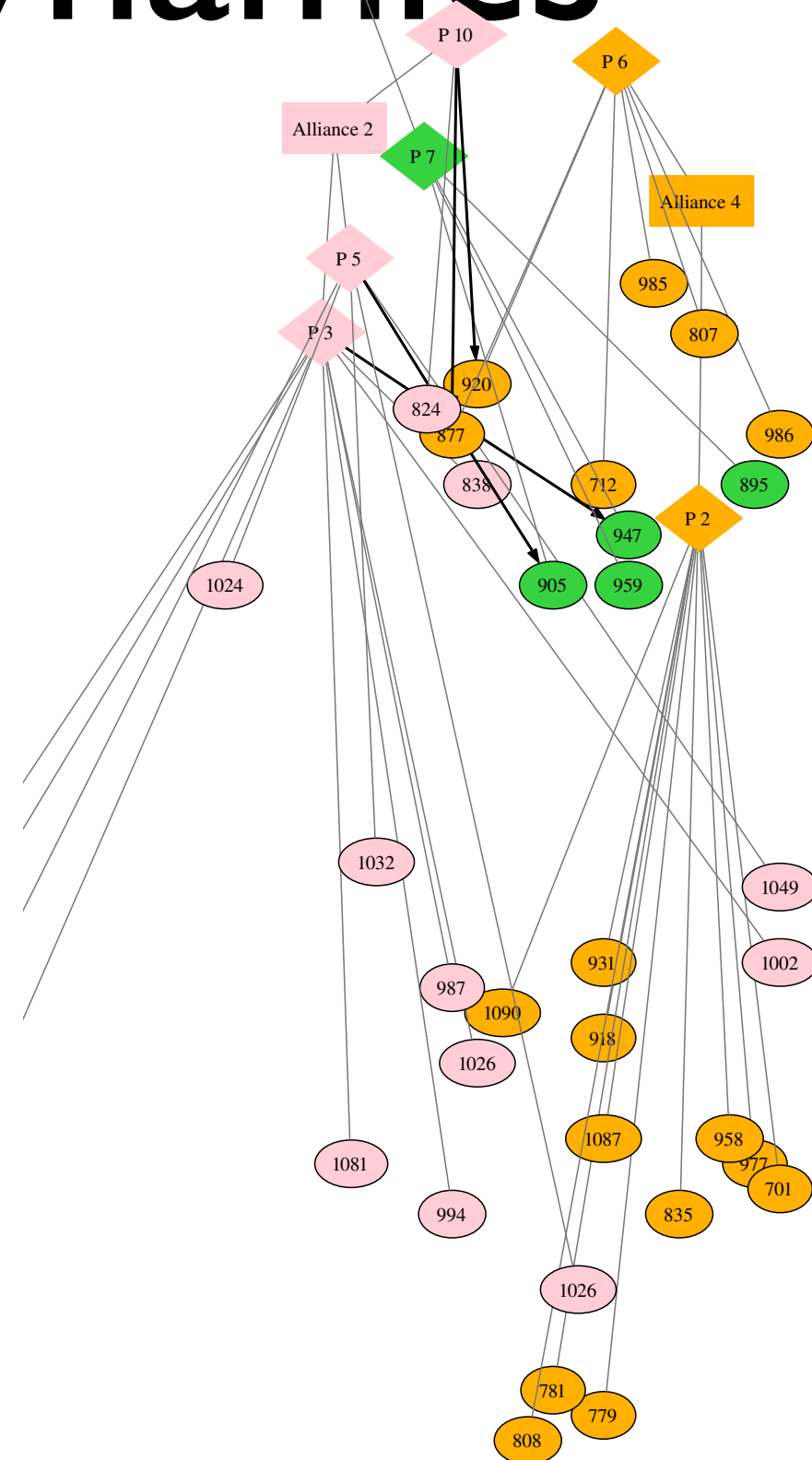
Fragment of world with

~10 alliances
~200 players
~600 cities

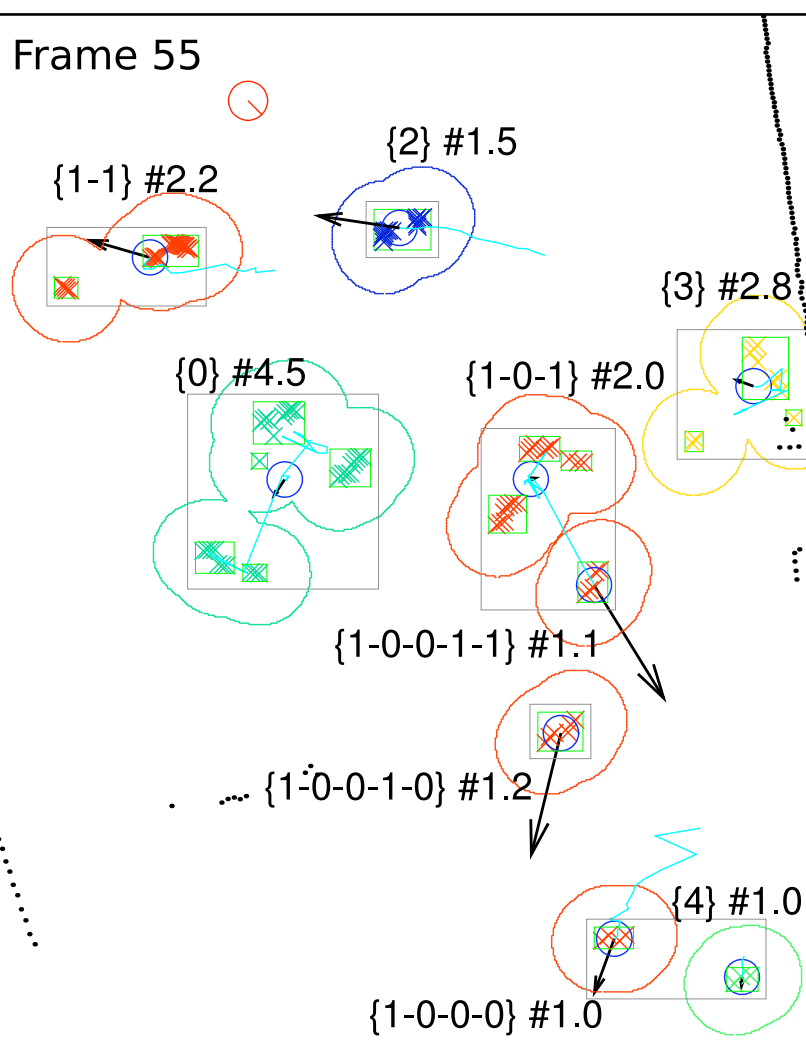
alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?

[Thon, Landwehr, De Raedt, ECML08]



Analyzing Video Data



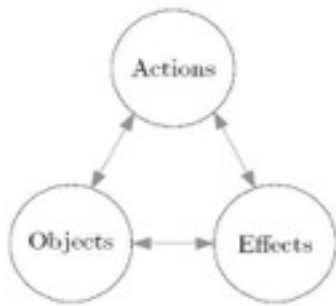
- Track people or objects over time? Even if temporarily hidden?

[Skarlatidis et al, TPLP 14;
Nitti et al, IROS 13, ICRA 14]

- Recognize activities?
- Infer object properties?

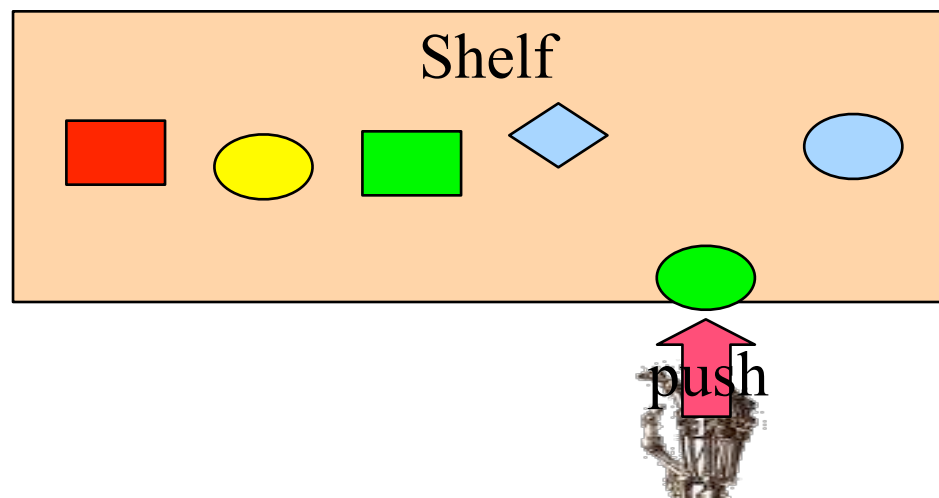
Learning relational affordances

Learn probabilistic model

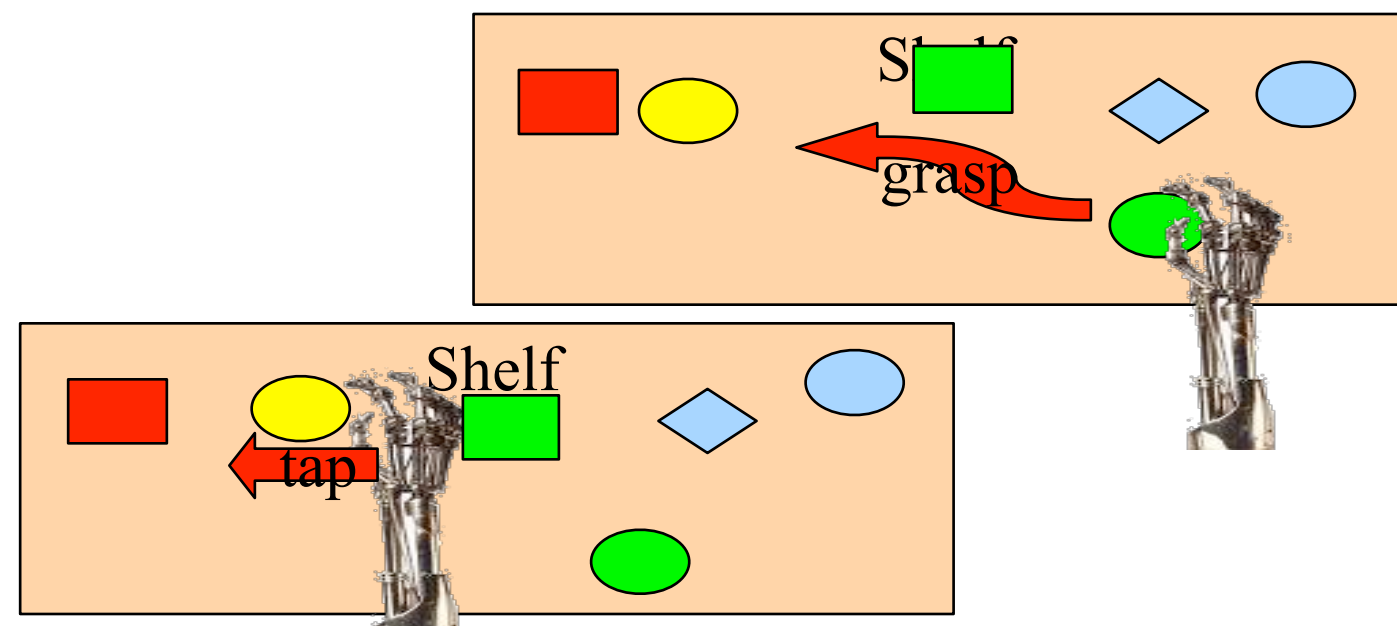


Inputs	Outputs	Function
(O, A)	E	Effect prediction
(O, E)	A	Action recognition/planning
(A, E)	O	Object recognition/selection

From two object interactions
Generalize to N



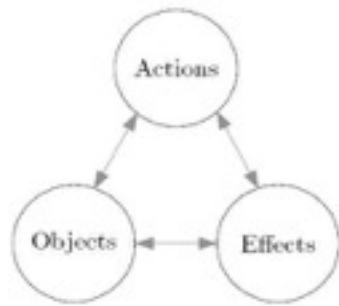
Moldovan et al. ICRA 12, 13, 14, PhD 15



Learning relational
affordances
between
two objects
(learnt by experience)

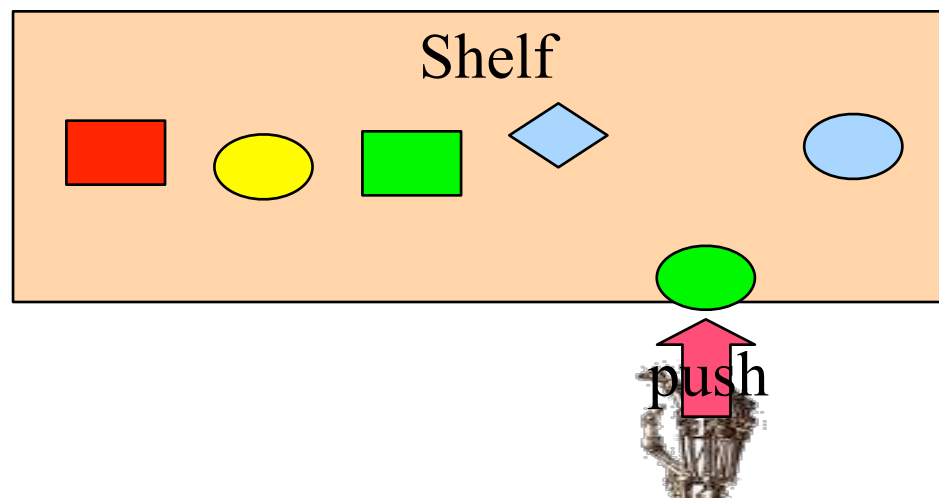
Learning relational affordances

Learn probabilistic model

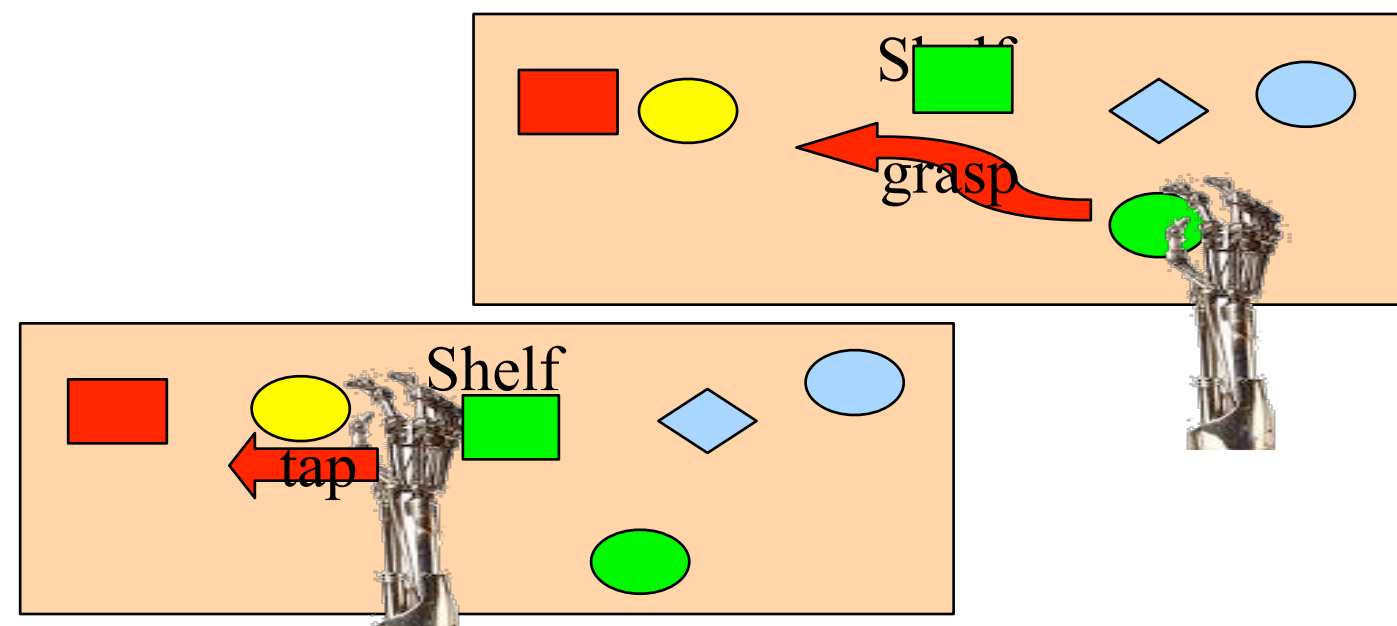


Inputs	Outputs	Function
(O, A)	E	Effect prediction
(O, E)	A	Action recognition/planning
(A, E)	O	Object recognition/selection

From two object interactions
Generalize to N



Moldovan et al. ICRA 12, 13, 14, PhD 15























Learning relational
affordances
between
two objects
(learnt by experience)

Example: Information Extraction


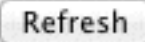




















Recently-Learned Facts

twitter

Refresh


instance	iteration	date learned	confidence
<u>kelly andrews</u> is a <u>female</u>	826	29-mar-2014	98.7  
<u>investment next year</u> is an <u>economic sector</u>	829	10-apr-2014	95.3  
<u>shibenik</u> is a <u>geopolitical entity</u> that is an organization	829	10-apr-2014	97.2  
<u>quality web design work</u> is a <u>character trait</u>	826	29-mar-2014	91.0  
<u>mercedes benz cls by carlsson</u> is an <u>automobile manufacturer</u>	829	10-apr-2014	95.2  
<u>social work</u> is an academic program <u>at the university rutgers university</u>	827	02-apr-2014	93.8  
<u>dante wrote</u> the book <u>the divine comedy</u>	826	29-mar-2014	93.8  
<u>willie aames</u> was <u>born in</u> the city <u>los angeles</u>	831	16-apr-2014	100.0  
<u>kitt peak</u> is a mountain <u>in the state or province arizona</u>	831	16-apr-2014	96.9  
<u>greenwich</u> is a park <u>in the city london</u>	831	16-apr-2014	100.0  





















Example: Information Extraction

Recently-Learned Facts 					
instance	iteration	date learned	confidence		
<u>kelly andrews</u> is a <u>female</u>	826	29-mar-2014	98.7		
<u>investment next year</u> is an <u>economic sector</u>	829	10-apr-2014	95.3		
<u>shibenik</u> is a <u>geopolitical entity</u> that is an organization	829	10-apr-2014	97.2		
<u>quality web design work</u> is a <u>character trait</u>	826	29-mar-2014	91.0		
<u>mercedes benz cls by carlsson</u> is an <u>automobile manufacturer</u>	829	10-apr-2014	95.2		
<u>social work</u> is an academic program <u>at the university rutgers university</u>	827	02-apr-2014	93.8		
<u>dante wrote</u> the book <u>the divine comedy</u>	826	29-mar-2014	93.8		
<u>willie aames</u> was <u>born in</u> the city <u>los angeles</u>	831	16-apr-2014	100.0		
<u>kitt peak</u> is a mountain <u>in the state or province arizona</u>	831	16-apr-2014	96.9		
<u>greenwich</u> is a park <u>in the city london</u>	831	16-apr-2014	100.0		

↑
instances for many
different relations

Example: Information Extraction

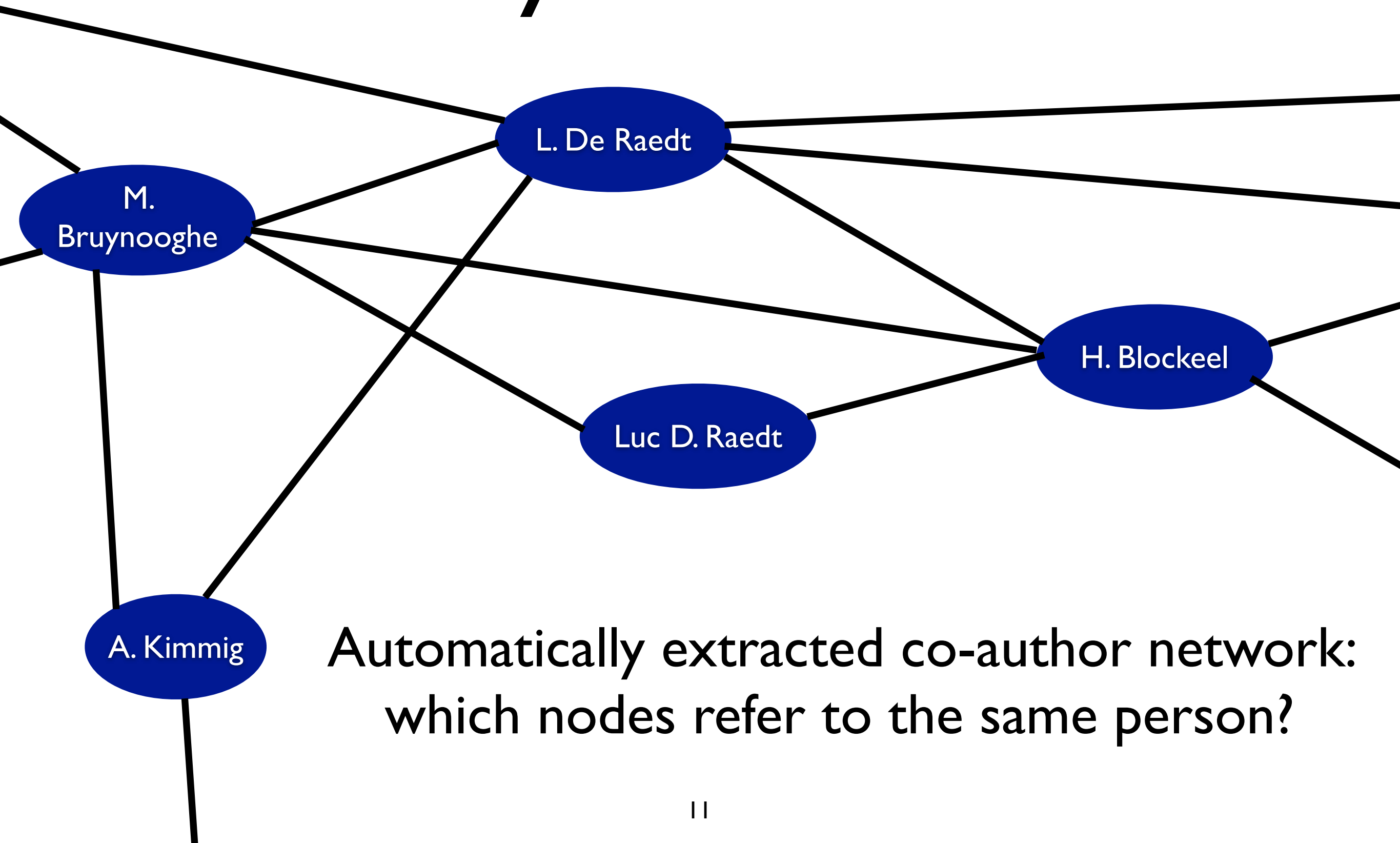
Recently-Learned Facts 

instance	iteration	date learned	confidence
<u>kelly andrews</u> is a <u>female</u>	826	29-mar-2014	98.7  
<u>investment next year</u> is an <u>economic sector</u>	829	10-apr-2014	95.3  
<u>shibenik</u> is a <u>geopolitical entity</u> that is an organization	829	10-apr-2014	97.2  
<u>quality web design work</u> is a <u>character trait</u>	826	29-mar-2014	91.0  
<u>mercedes benz cls by carlsson</u> is an <u>automobile manufacturer</u>	829	10-apr-2014	95.2  
<u>social work</u> is an academic program <u>at the university rutgers university</u>	827	02-apr-2014	93.8  
<u>dante wrote</u> the book <u>the divine comedy</u>	826	29-mar-2014	93.8  
<u>willie aames</u> was <u>born in</u> the city <u>los angeles</u>	831	16-apr-2014	100.0  
<u>kitt peak</u> is a mountain <u>in the state or province arizona</u>	831	16-apr-2014	96.9  
<u>greenwich</u> is a park <u>in the city london</u>	831	16-apr-2014	100.0  

↑
instances for many
different relations

↑
degree of certainty

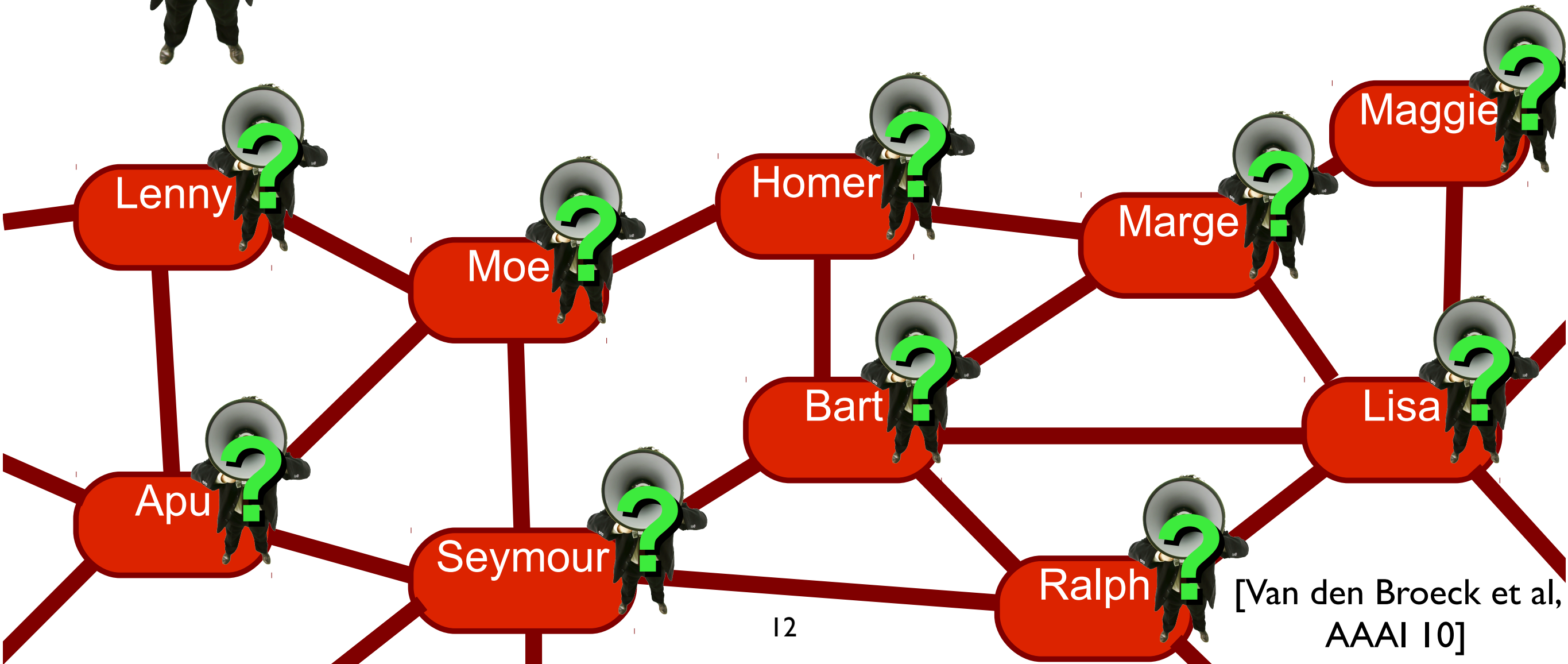
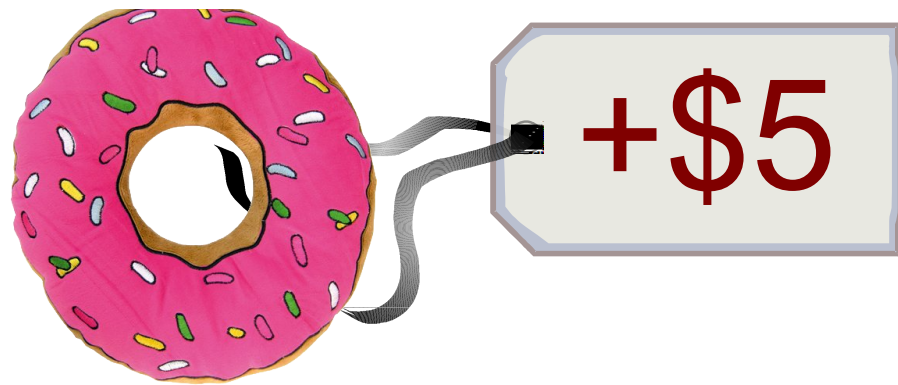
Entity Resolution



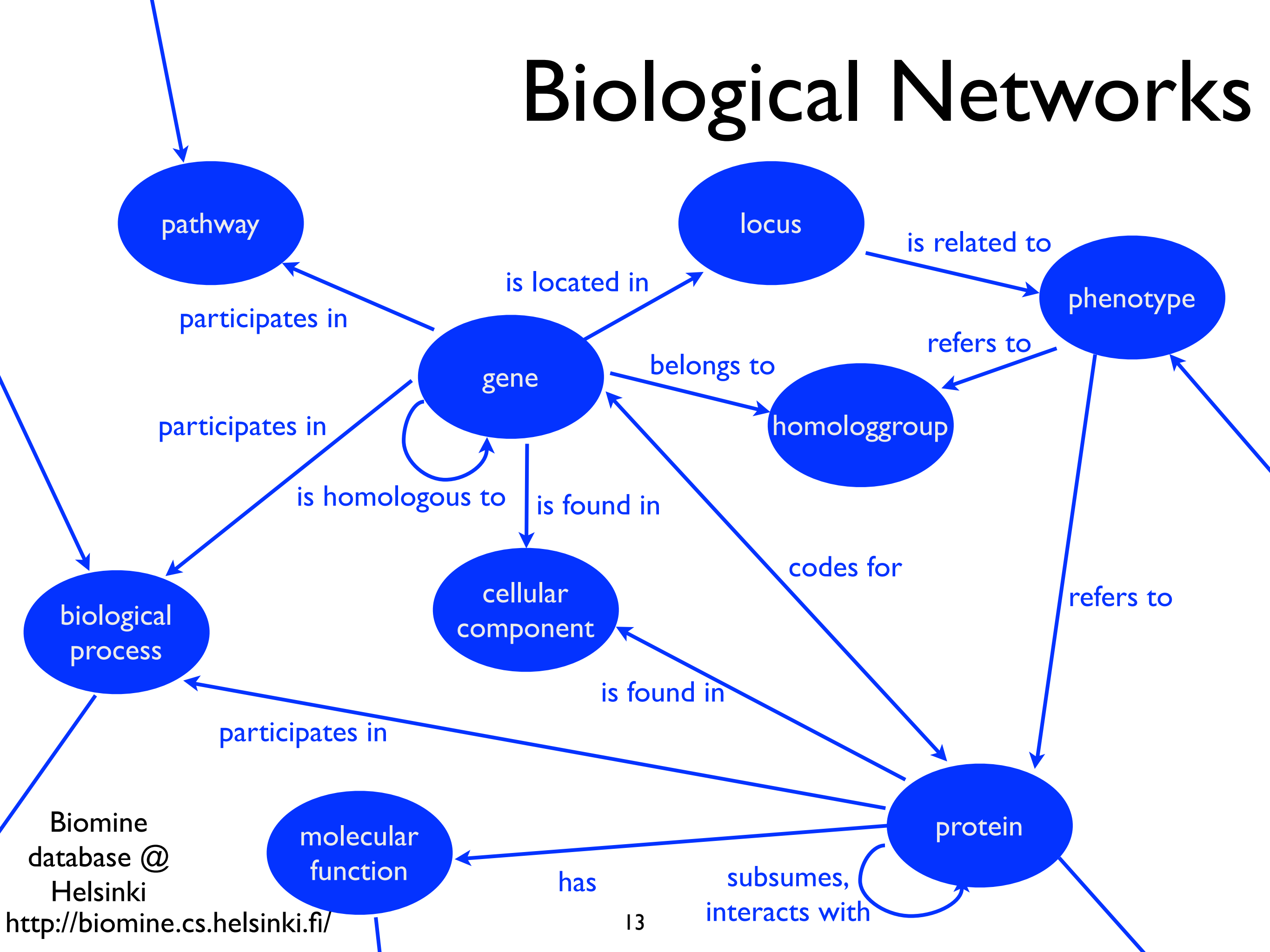
Automatically extracted co-author network:
which nodes refer to the same person?

Viral Marketing

Which advertising strategy maximizes expected profit?



Biological Networks

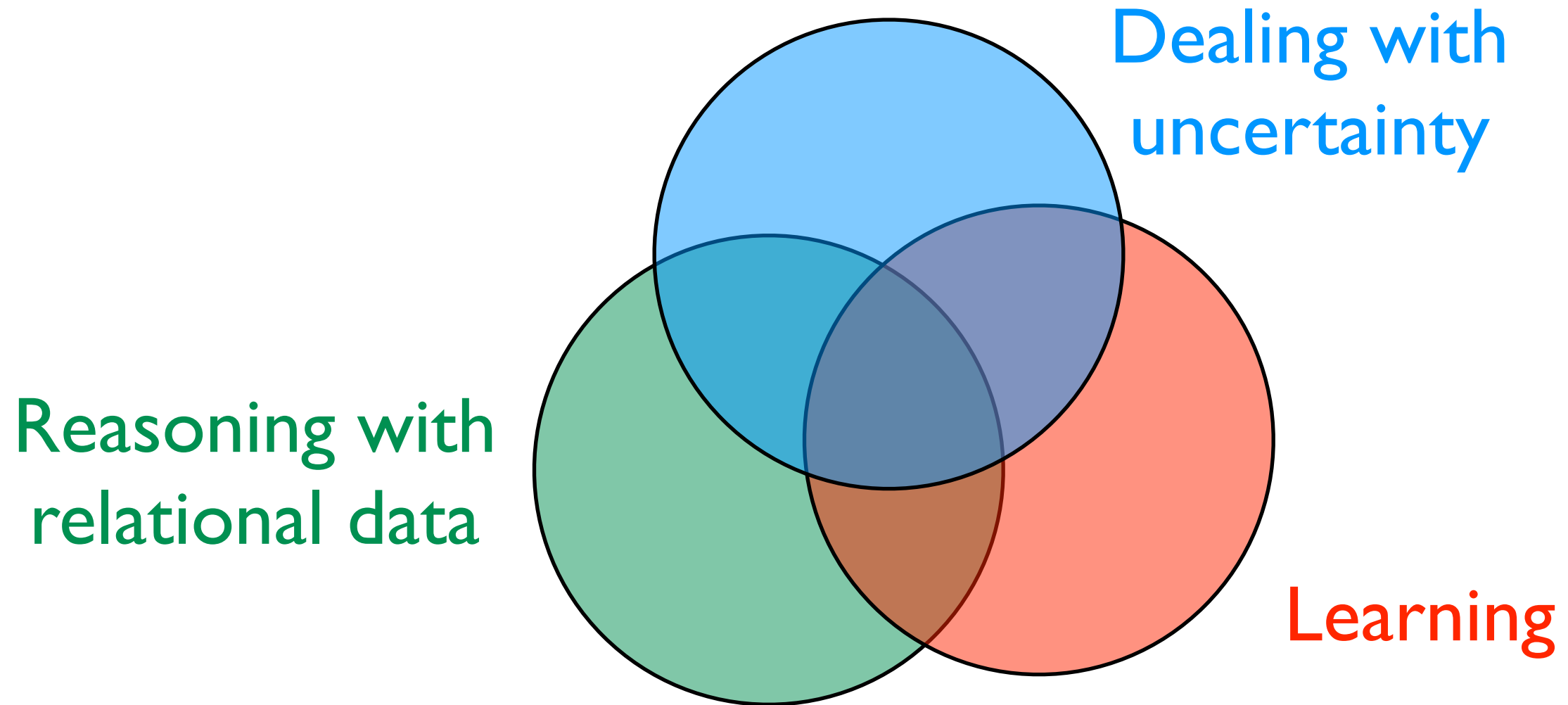


This requires dealing with

- Structured environments
 - objects, and
 - **relationships** amongst them
- and possibly
 - using background knowledge
- cope with **uncertainty**
- **learn from data**

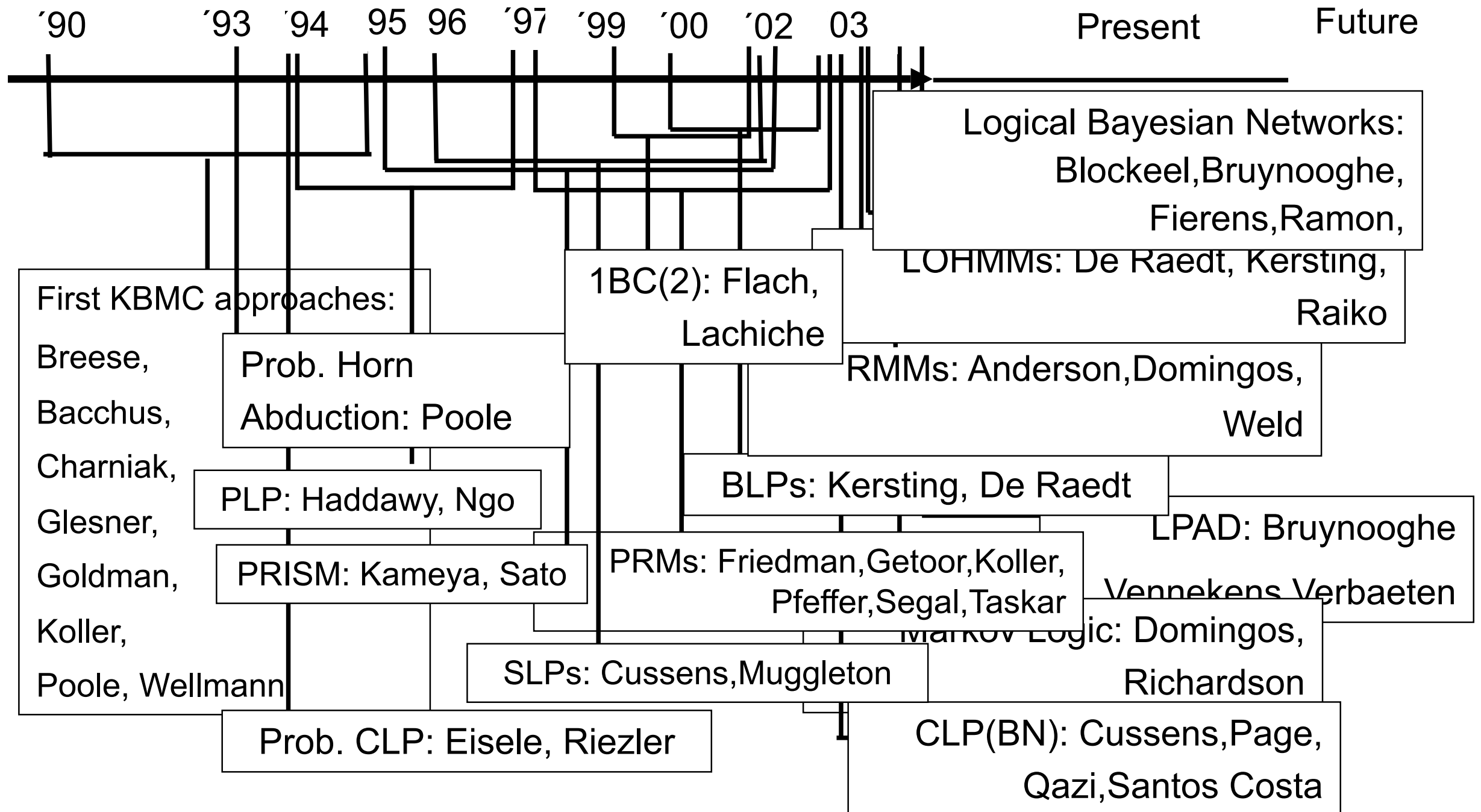
Statistical Relational Learning
Probabilistic Programming

Common theme

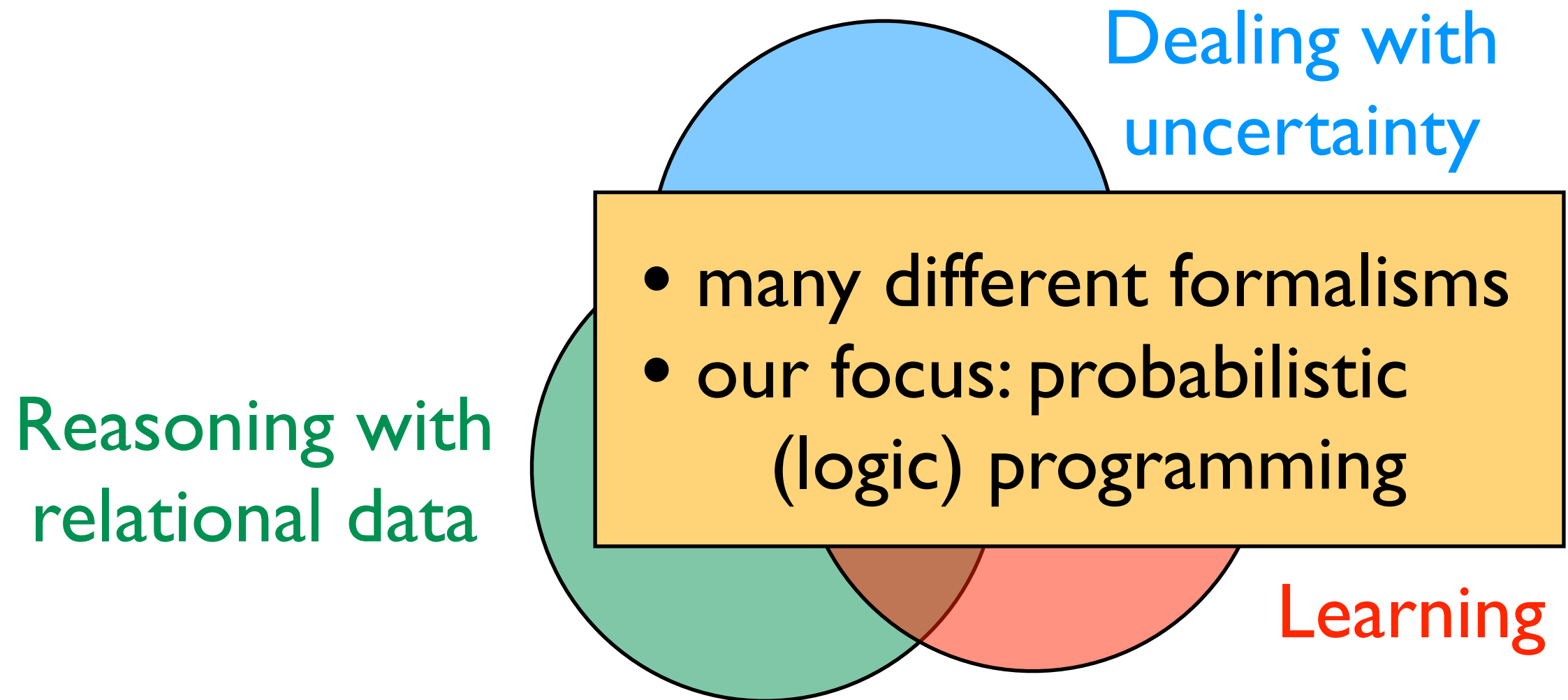


Statistical relational learning
& Probabilistic Programming, ...

Some formalisms



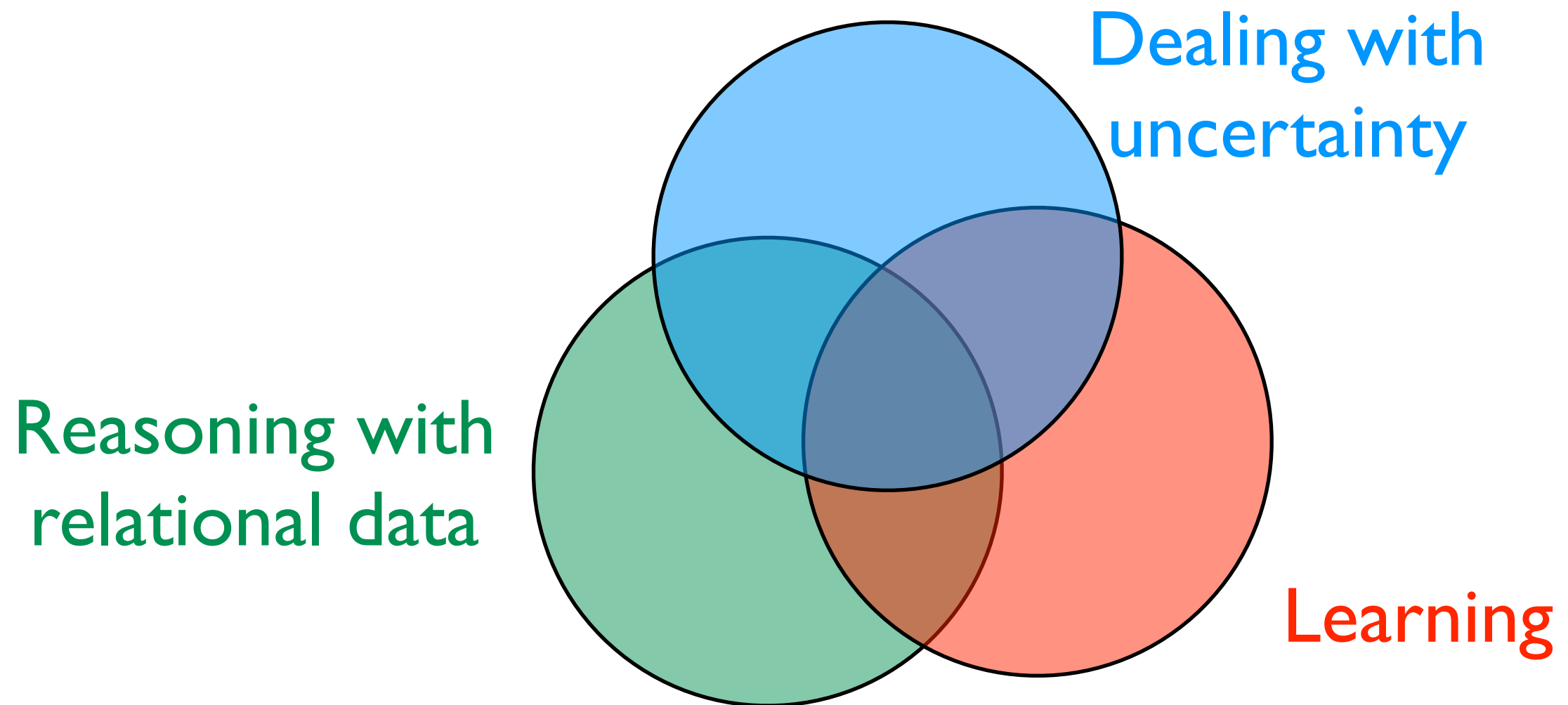
Common theme



Statistical relational learning
& Probabilistic Programming, ...

ProbLog

probabilistic Prolog



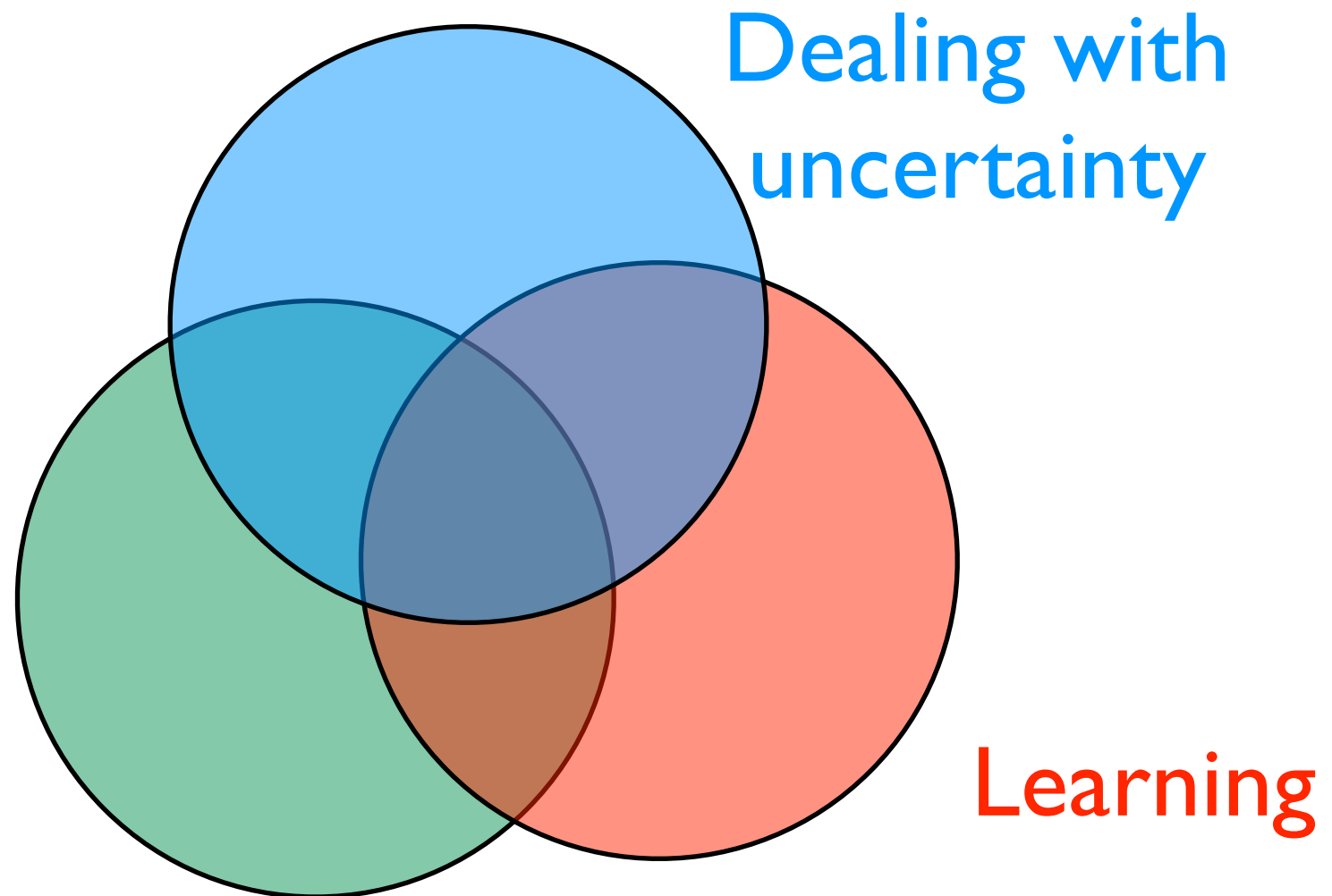
ProbLog

probabilistic Prolog

Prolog / logic
programming

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) , smokes(Y) .
```



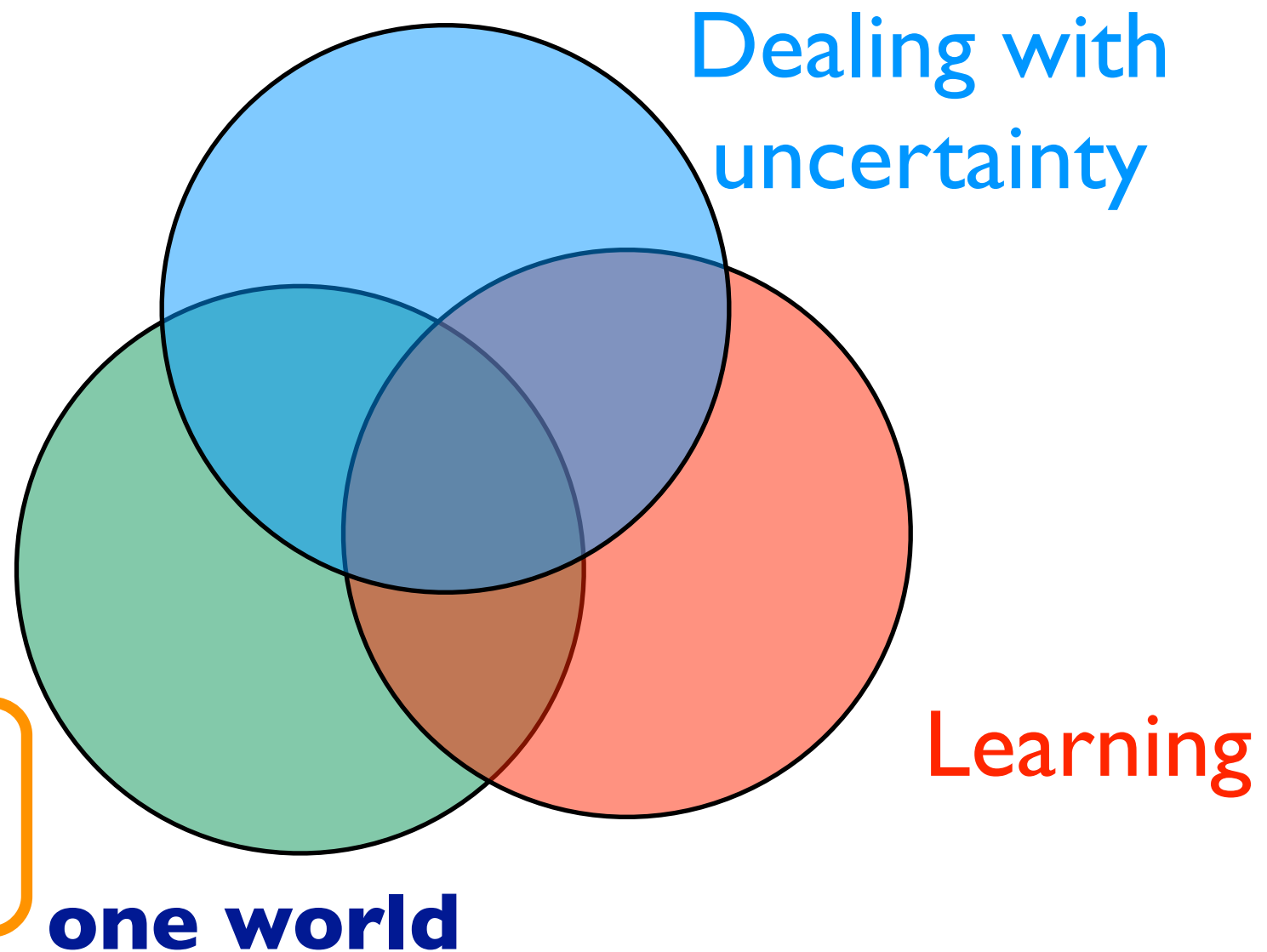
ProbLog

probabilistic Prolog

Prolog / logic
programming

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) , smokes(Y) .
```



ProbLog

probabilistic Prolog

```
0.8::stress(ann).  
0.6::influences(ann,bob).  
0.2::influences(bob,carl).
```

atoms as random
variables

Prolog / logic
programming

```
stress(ann).  
influences(ann,bob).  
influences(bob,carl).
```

one world

Learning

```
smokes(X) :- stress(X).  
smokes(X) :-  
    influences(Y,X), smokes(Y).
```


ProbLog

probabilistic Prolog

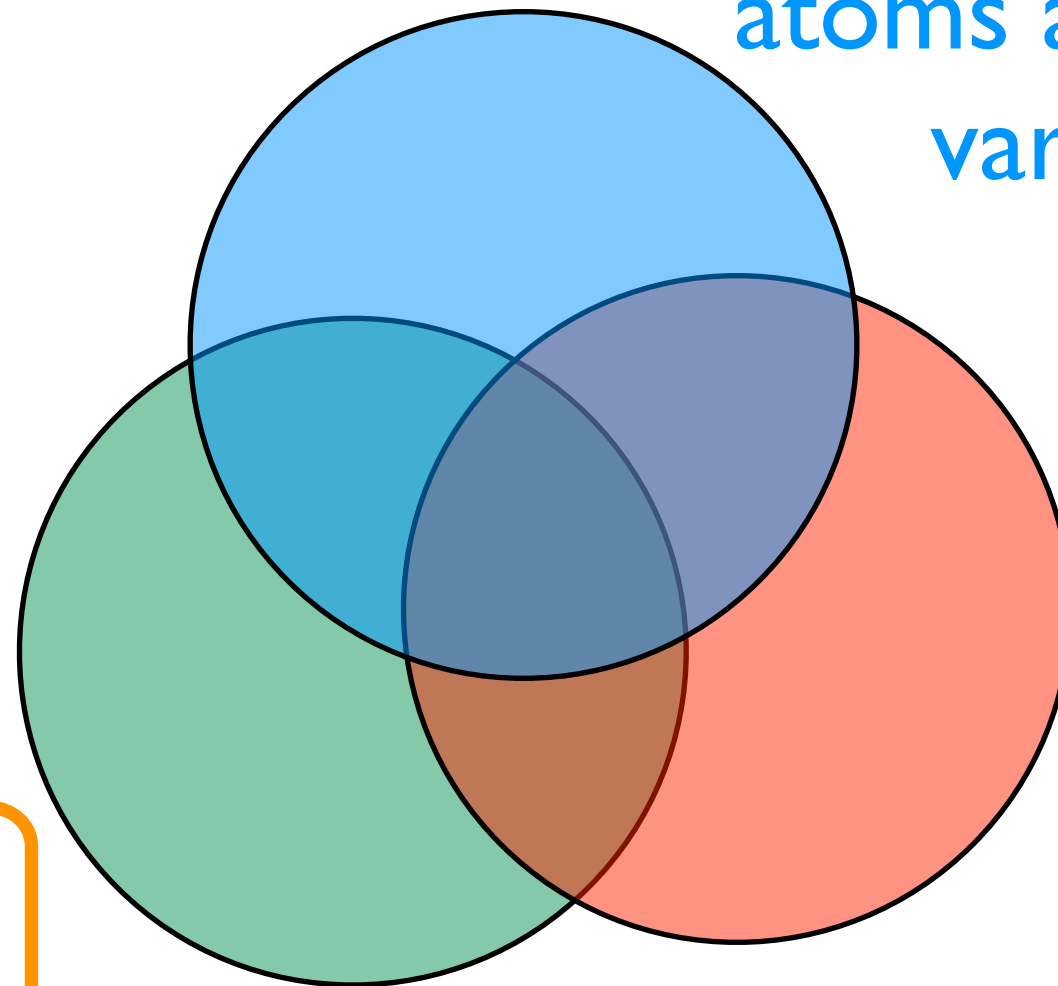
several possible worlds

```
0.8::stress(ann).  
0.6::influences(ann,bob).  
0.2::influences(bob,carl).
```

atoms as random
variables

Prolog / logic
programming

```
stress(ann).  
influences(ann,bob).  
influences(bob,carl).
```



Learning

one world

```
smokes(X) :- stress(X).  
smokes(X) :-  
    influences(Y,X), smokes(Y).
```

ProbLog

probabilistic Prolog

several possible worlds

```
0.8::stress(ann).  
0.6::influences(ann,bob).  
0.2::influences(bob,carl).
```

atoms as random
variables

Prolog / logic
programming

```
stress(ann).  
influences(ann,bob).  
influences(bob,carl).
```

one world

```
smokes(X) :- stress(X).  
smokes(X) :-  
    influences(Y,X), smokes(Y).
```

parameter learning,
adapted relational
learning techniques

Probabilistic Logic Programming

Distribution Semantics [Sato, ICLP 95]:
probabilistic choices + logic program
→ distribution over possible worlds

e.g., PRISM, ICL, ProbLog, LPADs, CP-logic, ...

multi-valued
switches

probabilistic
facts

probabilistic
alternatives

annotated
disjunctions

causal-
probabilistic
laws

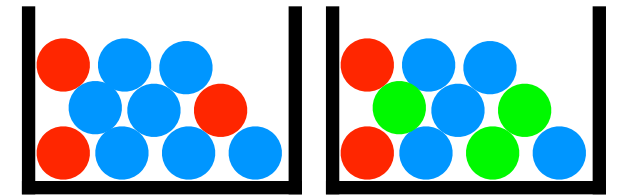
Roadmap

- Modeling (ProbLog and Church, another representative of PP)
- Inference
- Learning
- Dynamics and Decisions

... with some detours on the way

ProbLog by example:

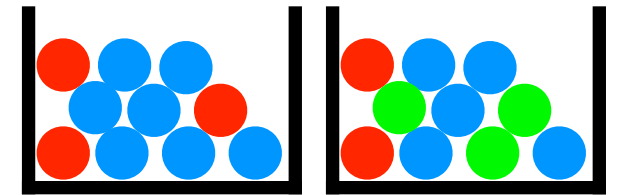
A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

ProbLog by example:

A bit of gambling



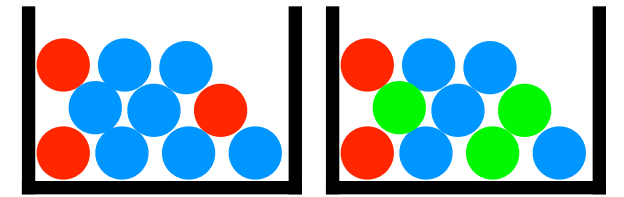
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

0.4 :: heads.

probabilistic fact: heads is true with probability 0.4 (and false with 0.6)

ProbLog by example:

A bit of gambling

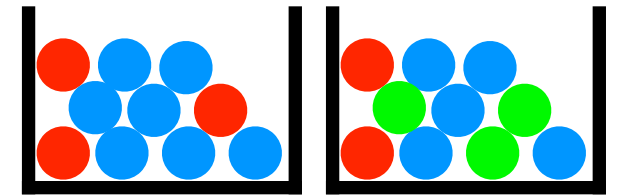


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

0.4 :: heads . **annotated disjunction:** first ball is red
with probability 0.3 and blue with 0.7

0.3 :: col(1,red) ; 0.7 :: col(1,blue) .

ProbLog by example:



A bit of gambling

- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

`0.4 :: heads.`

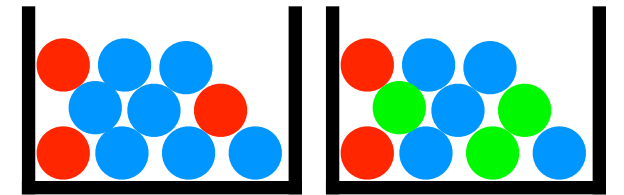
`0.3 :: col(1,red) ; 0.7 :: col(1,blue) .`

`0.2 :: col(2,red) ; 0.3 :: col(2,green) ;`

`0.5 :: col(2,blue) .`

annotated disjunction: second ball is red with probability 0.2, green with 0.3, and blue with 0.5

ProbLog by example:



A bit of gambling

- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

`0.4 :: heads.`

`0.3 :: col(1,red) ; 0.7 :: col(1,blue) .`

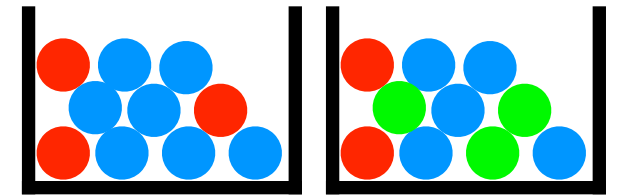
`0.2 :: col(2,red) ; 0.3 :: col(2,green) ;`

`0.5 :: col(2,blue) .`

`win :- heads, col(_,red) .`

logical rule encoding
background knowledge

ProbLog by example:



A bit of gambling

- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

`0.4 :: heads.`

`0.3 :: col(1,red) ; 0.7 :: col(1,blue) .`

`0.2 :: col(2,red) ; 0.3 :: col(2,green) ;`

`0.5 :: col(2,blue) .`

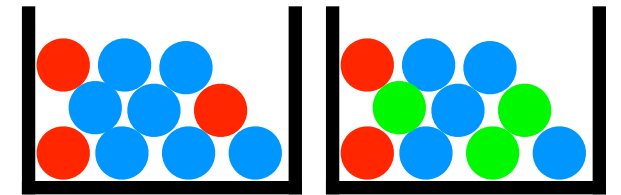
`win :- heads, col(_,red) .`

`win :- col(1,C) , col(2,C) .`

logical rule encoding
background knowledge

ProbLog by example:

A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.
```

probabilistic choices

```
0.3 :: col(1,red) ; 0.7 :: col(1,blue) .
```

```
0.2 :: col(2,red) ; 0.3 :: col(2,green) ;
```

```
0.5 :: col(2,blue) .
```

```
win :- heads, col(_,red) .
```

```
win :- col(1,C) , col(2,C) .
```

consequences

Questions

`0.4 :: heads.`

`0.3 :: col(1,red) ; 0.7 :: col(1,blue).`

`0.2 :: col(2,red) ; 0.3 :: col(2,green) ; 0.5 :: col(2,blue).`

`win :- heads, col(_,red).`

`win :- col(1,C), col(2,C).`

- Probability of **win**?
- Probability of **win** given `col(2,green)`?
- Most probable world where **win** is true?

Questions

```
0.4 :: heads.
```

```
0.3 :: col(1,red) ; 0.7 :: col(1,blue).
```

```
0.2 :: col(2,red) ; 0.3 :: col(2,green) ; 0.5 :: col(2,blue).
```

```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

marginal probability

- Probability of **win**
query
- Probability of **win** given **col(2,green)**?
- Most probable world where **win** is true?

Questions

`0.4 :: heads.`

`0.3 :: col(1,red) ; 0.7 :: col(1,blue).`

`0.2 :: col(2,red) ; 0.3 :: col(2,green) ; 0.5 :: col(2,blue).`

`win :- heads, col(_,red).`

`win :- col(1,C), col(2,C).`

marginal probability

- Probability of `win`?

conditional probability

- Probability of `win` given `col(2,green)`?

evidence

- Most probable world where `win` is true?

Questions

0.4 :: heads.

0.3 :: col(1,red) ; 0.7 :: col(1,blue).

0.2 :: col(2,red) ; 0.3 :: col(2,green) ; 0.5 :: col(2,blue).

win :- heads, col(_,red).

win :- col(1,C), col(2,C).

marginal probability

- Probability of **win**?

conditional probability

- Probability of **win** given col(2,green)?

- Most probable world where **win** is true?

MPE inference

Possible Worlds

0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).

0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).

win :- col(1,C), col(2,C).

Possible Worlds

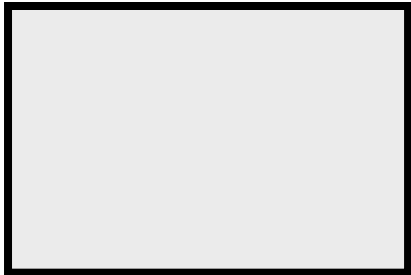
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).

0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).

win :- col(1,C), col(2,C).



Possible Worlds

`0.4 :: heads.`

`0.3 :: col(1,red) ; 0.7 :: col(1,blue) .`

`0.2 :: col(2,red) ; 0.3 :: col(2,green) ; 0.5 :: col(2,blue) .`

`win :- heads, col(_,red) .`

`win :- col(1,C) , col(2,C) .`

0.4



Possible Worlds

```
0.4 :: heads.
```

```
0.3 :: col(1,red); 0.7 :: col(1,blue).
```

```
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).
```

```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

0.4×0.3



Possible Worlds

```
0.4 :: heads.
```

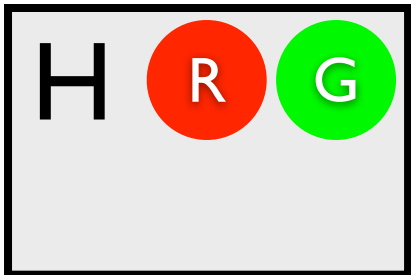
```
0.3 :: col(1,red); 0.7 :: col(1,blue).
```

```
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).
```

```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$



Possible Worlds

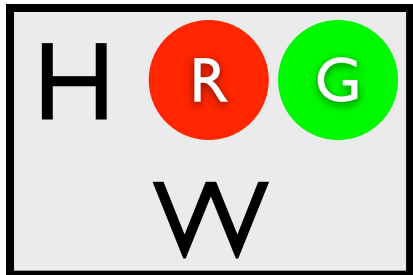
`0.4 :: heads.`

`0.3 :: col(1,red); 0.7 :: col(1,blue).`

`0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).`

```
win :- heads, col(_,red).  
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$



Possible Worlds

`0.4 :: heads.`

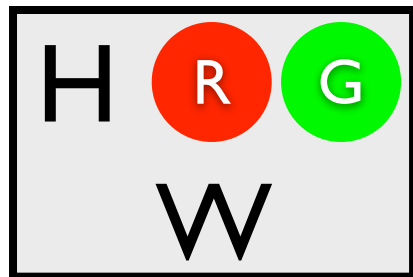
`0.3 :: col(1,red) ; 0.7 :: col(1,blue) .`

`0.2 :: col(2,red) ; 0.3 :: col(2,green) ; 0.5 :: col(2,blue) .`

`win :- heads, col(_,red) .`

`win :- col(1,C), col(2,C) .`

$0.4 \times 0.3 \times 0.3$



$(1 - 0.4)$



Possible Worlds

0.4 :: heads.

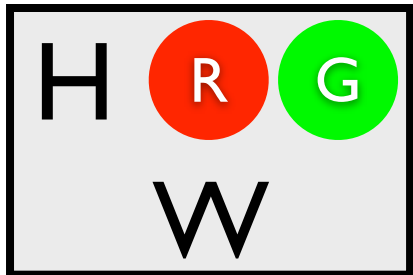
0.3 :: col(1,red); 0.7 :: col(1,blue).

0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

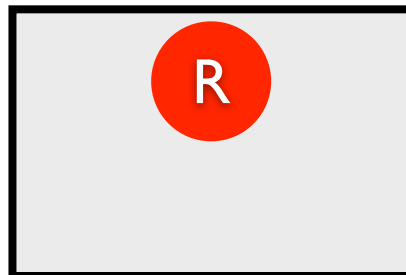
win :- heads, col(_,red).

win :- col(1,C), col(2,C).

$$0.4 \times 0.3 \times 0.3$$



$$(1 - 0.4) \times 0.3$$



Possible Worlds

```
0.4 :: heads.
```

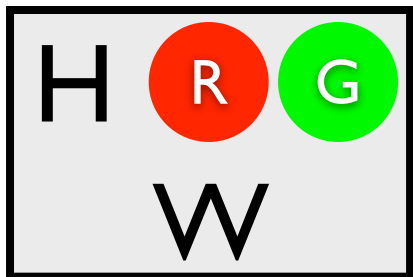
```
0.3 :: col(1,red); 0.7 :: col(1,blue).
```

```
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).
```

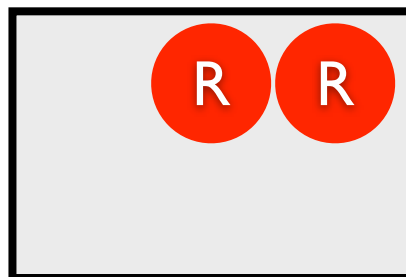
```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

$$0.4 \times 0.3 \times 0.3$$



$$(1-0.4) \times 0.3 \times 0.2$$



Possible Worlds

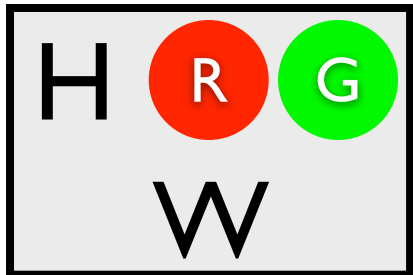
`0.4 :: heads.`

`0.3 :: col(1,red); 0.7 :: col(1,blue).`

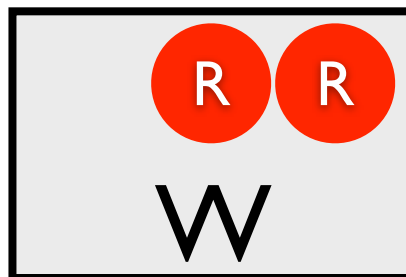
`0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).`

`win :- heads, col(_,red).`
`win :- col(1,C), col(2,C).`

$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$



Possible Worlds

`0.4 :: heads.`

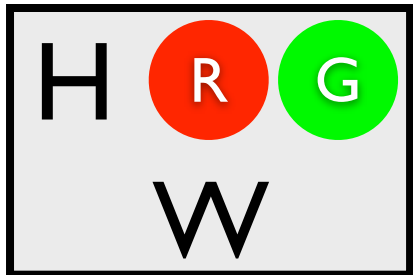
`0.3 :: col(1,red) ; 0.7 :: col(1,blue) .`

`0.2 :: col(2,red) ; 0.3 :: col(2,green) ; 0.5 :: col(2,blue) .`

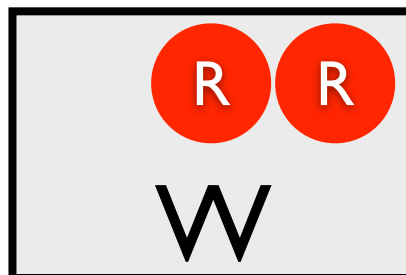
`win :- heads, col(_,red) .`

`win :- col(1,C), col(2,C) .`

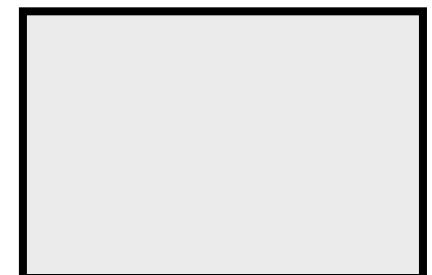
$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$



$(1-0.4)$



Possible Worlds

`0.4 :: heads.`

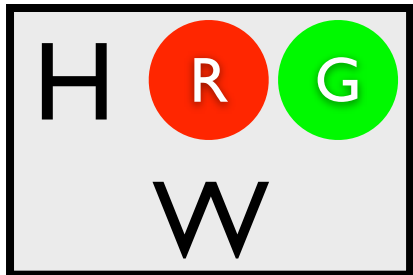
`0.3 :: col(1,red); 0.7 :: col(1,blue).`

`0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).`

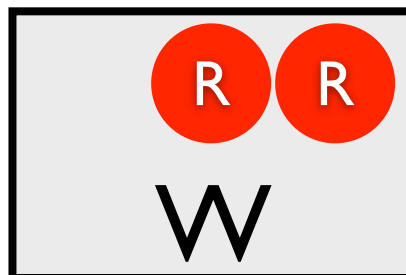
`win :- heads, col(_,red).`

`win :- col(1,C), col(2,C).`

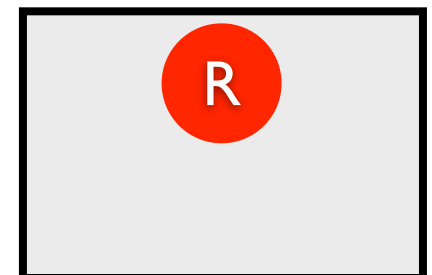
$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$



$(1-0.4) \times 0.3$



Possible Worlds

```
0.4 :: heads.
```

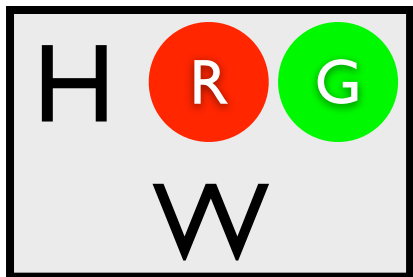
```
0.3 :: col(1,red); 0.7 :: col(1,blue)
```

```
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).
```

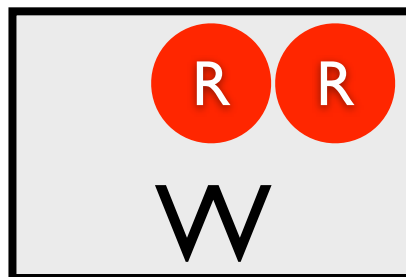
```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

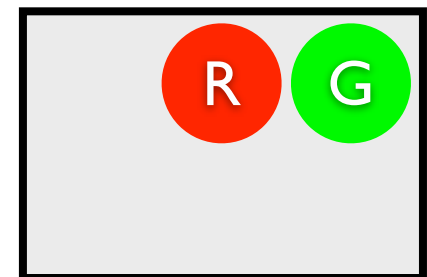
$$0.4 \times 0.3 \times 0.3$$



$$(1-0.4) \times 0.3 \times 0.2$$



$$(1-0.4) \times 0.3 \times 0.3$$



Possible Worlds

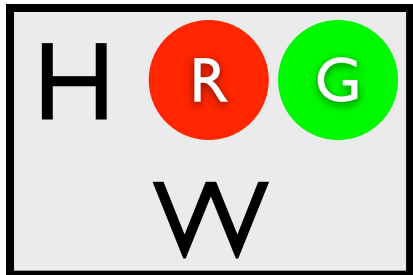
`0.4 :: heads.`

`0.3 :: col(1,red); 0.7 :: col(1,blue).`

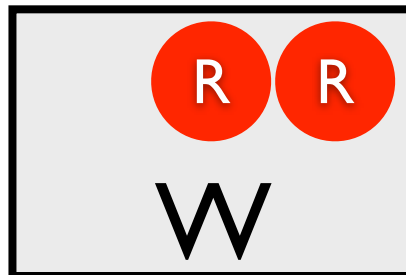
`0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).`

```
win :- heads, col(_,red).  
win :- col(1,C), col(2,C).
```

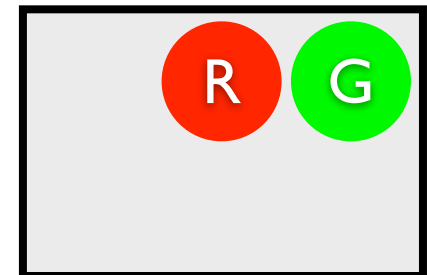
$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$



$(1-0.4) \times 0.3 \times 0.3$



Possible Worlds

```
0.4 :: heads.
```

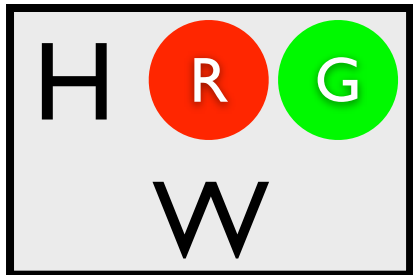
```
0.3 :: col(1,red); 0.7 :: col(1,blue).
```

```
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).
```

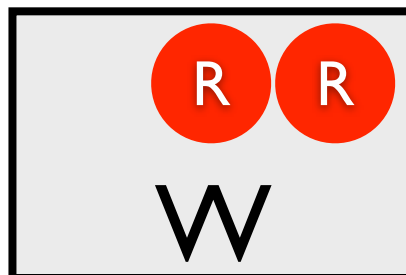
```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

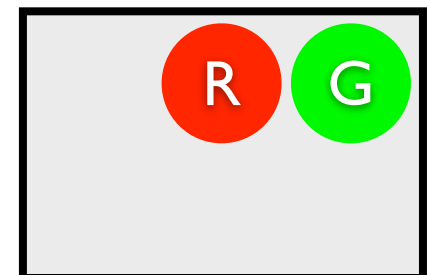
$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$

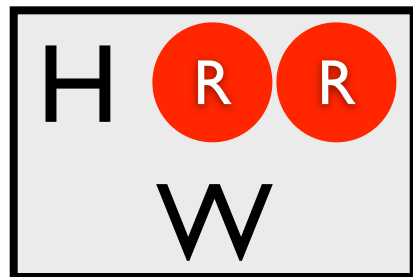


$(1-0.4) \times 0.3 \times 0.3$

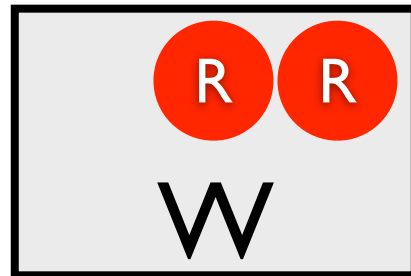


All Possible Worlds

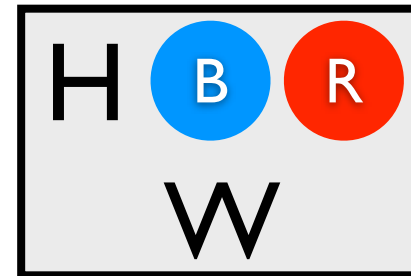
0.024



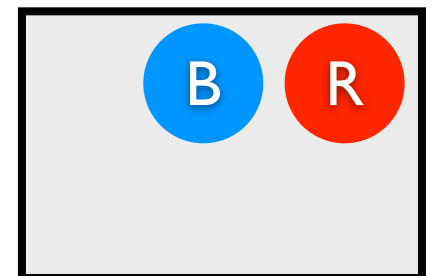
0.036



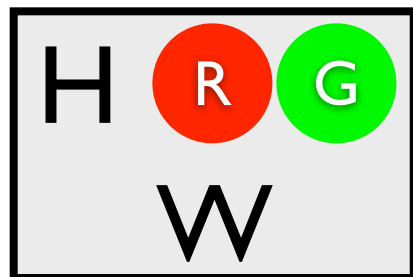
0.056



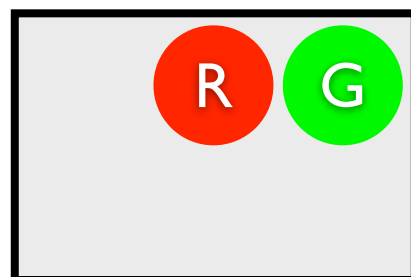
0.084



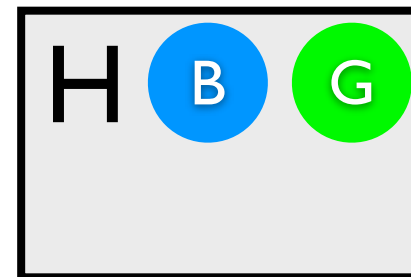
0.036



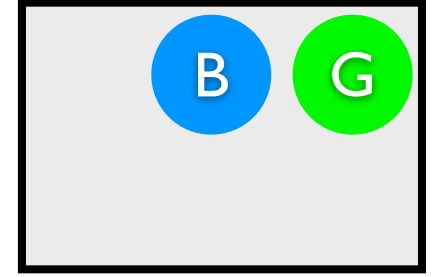
0.054



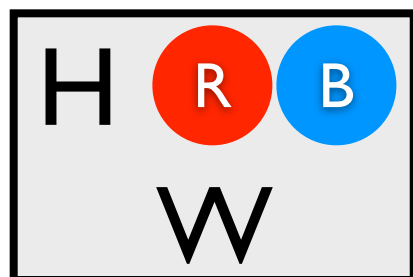
0.084



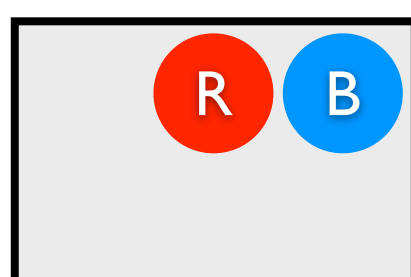
0.126



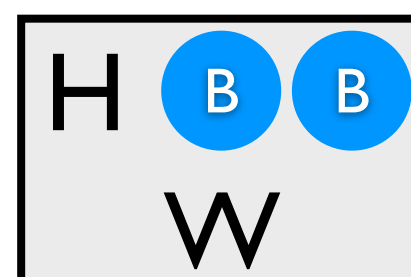
0.060



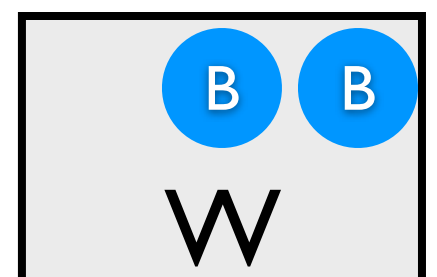
0.090



0.140



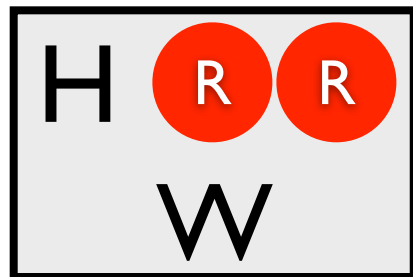
0.210



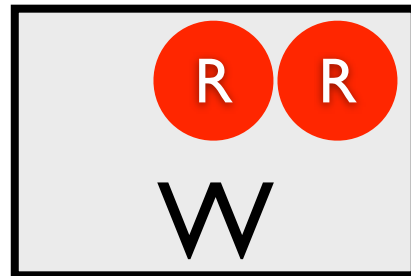
Most likely world where `win` is true?

MPE Inference

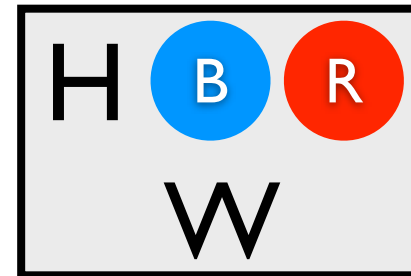
0.024



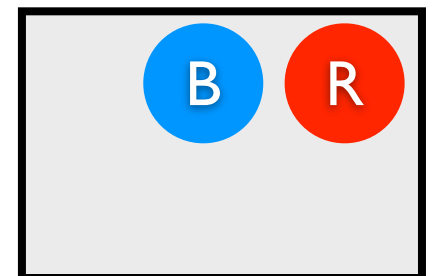
0.036



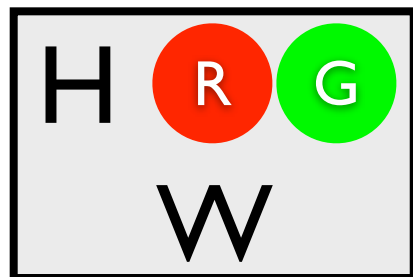
0.056



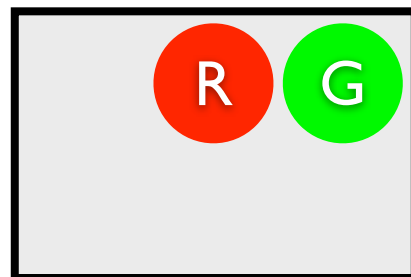
0.084



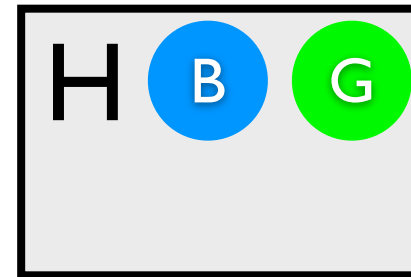
0.036



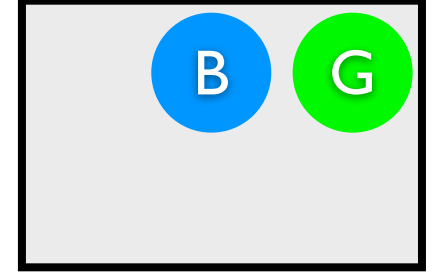
0.054



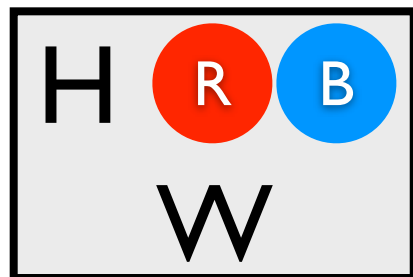
0.084



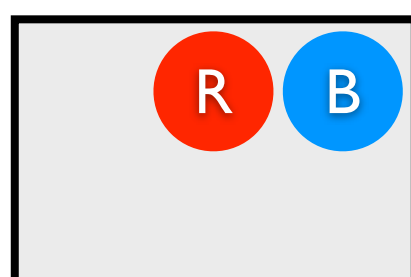
0.126



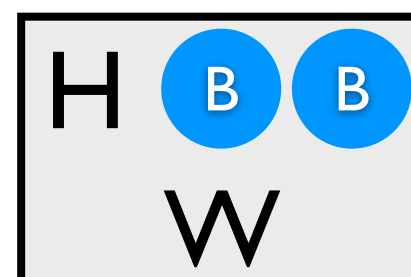
0.060



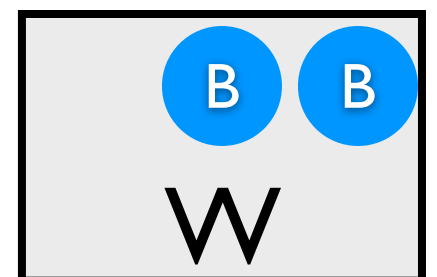
0.090



0.140



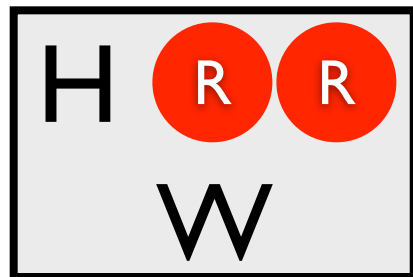
0.210



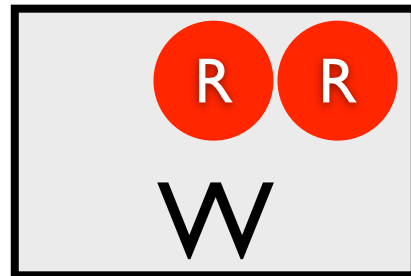
Most likely world where `win` is true?

MPE Inference

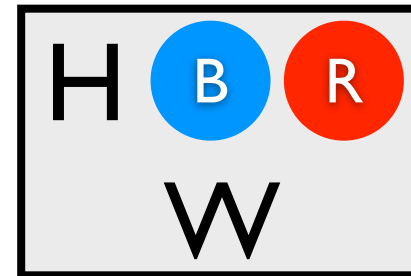
0.024



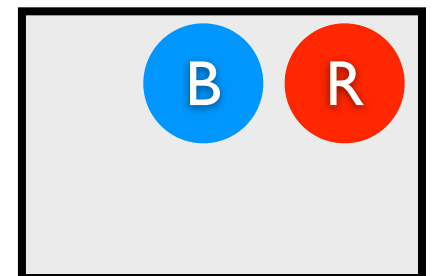
0.036



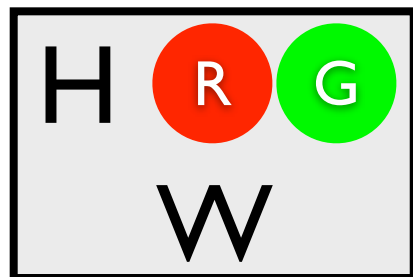
0.056



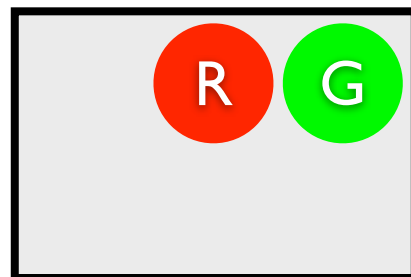
0.084



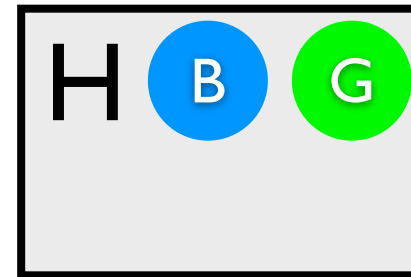
0.036



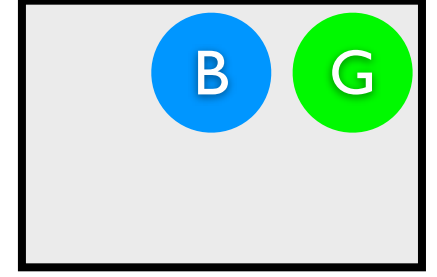
0.054



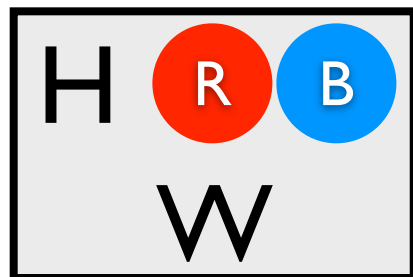
0.084



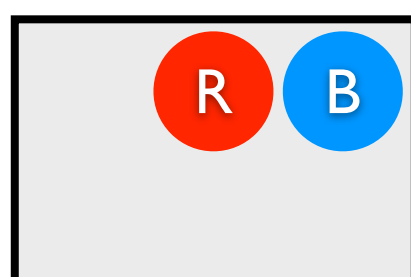
0.126



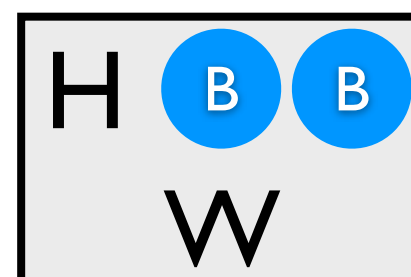
0.060



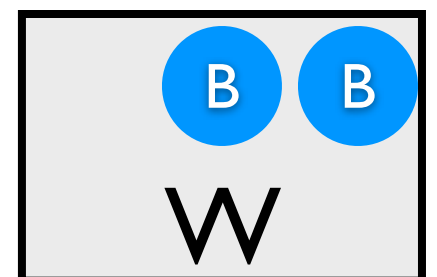
0.090



0.140



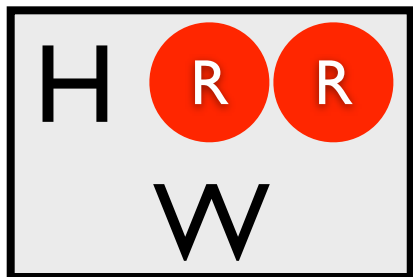
0.210



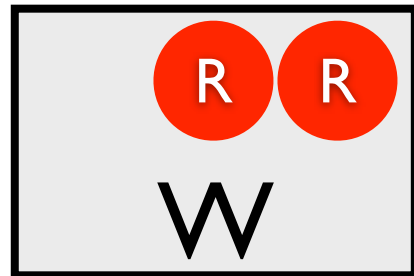
Most likely world where `col(2, blue)` is false?

MPE Inference

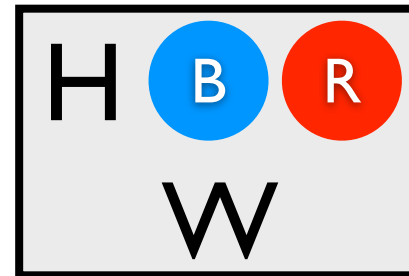
0.024



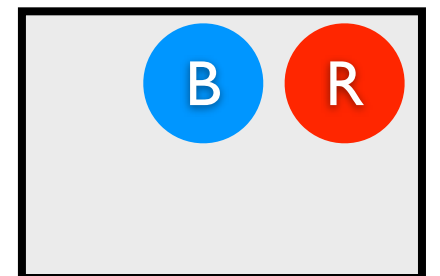
0.036



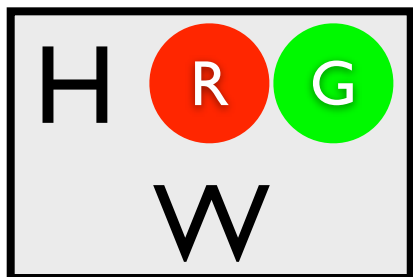
0.056



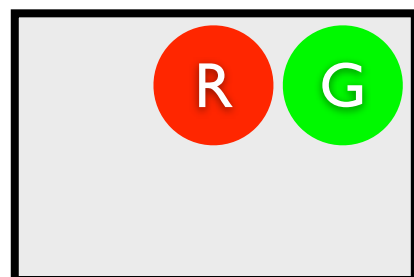
0.084



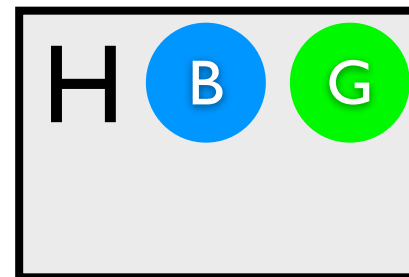
0.036



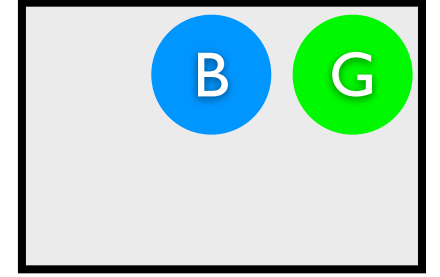
0.054



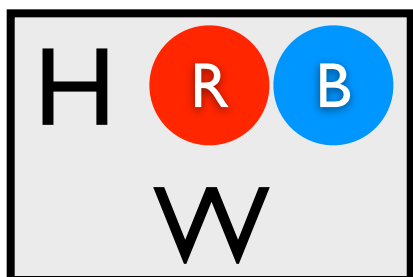
0.084



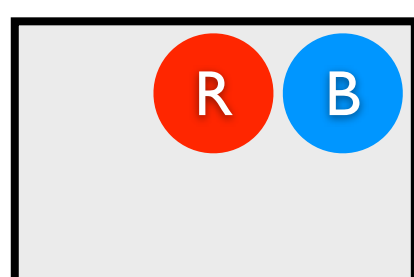
0.126



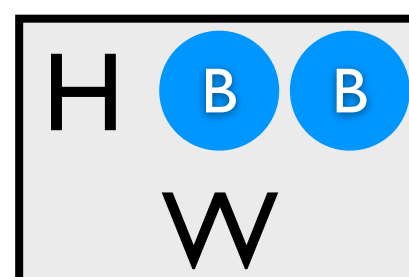
0.060



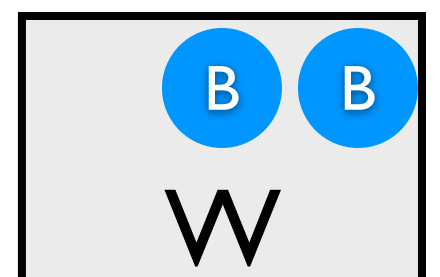
0.090



0.140



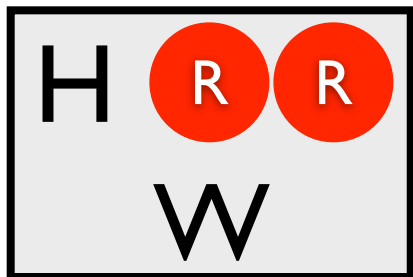
0.210



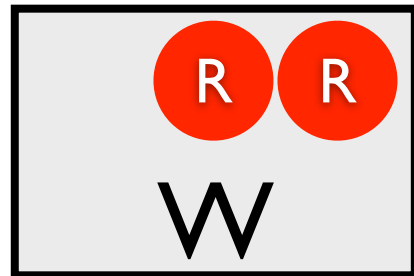
Most likely world where `col(2, blue)` is false?

MPE Inference

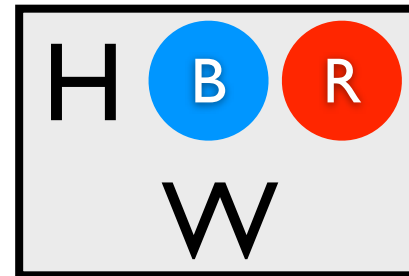
0.024



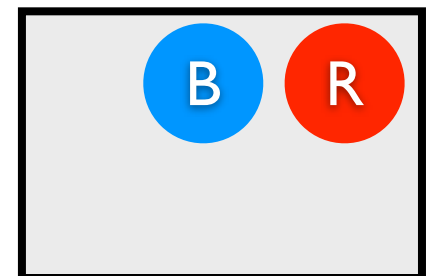
0.036



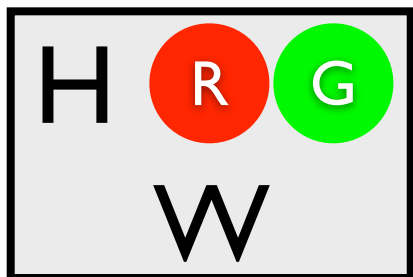
0.056



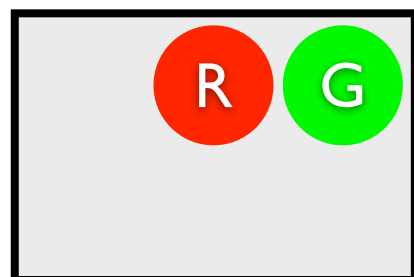
0.084



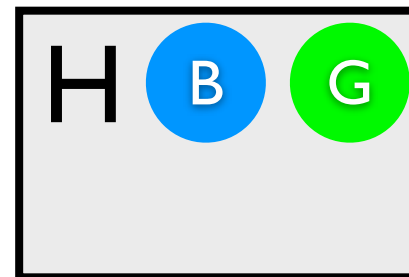
0.036



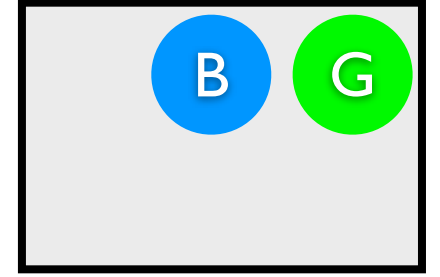
0.054



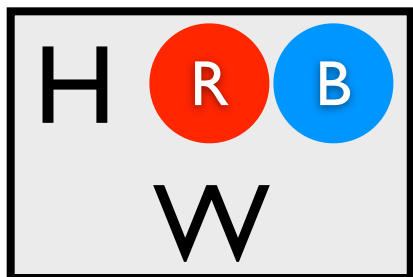
0.084



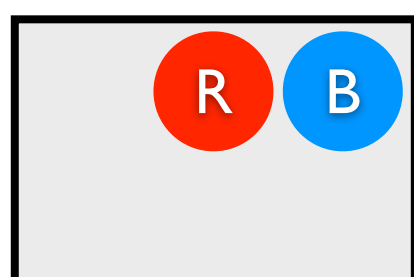
0.126



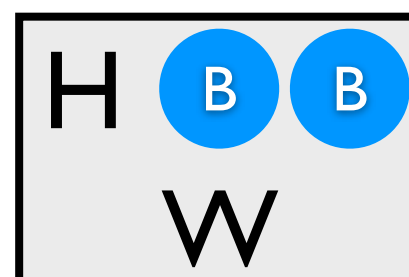
0.060



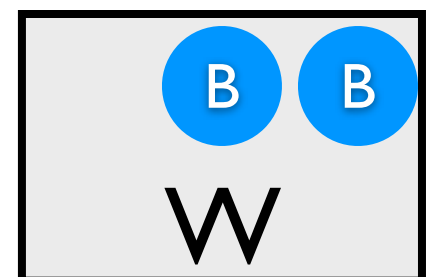
0.090



0.140



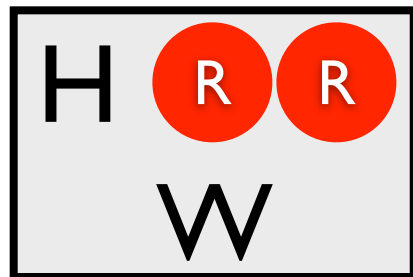
0.210



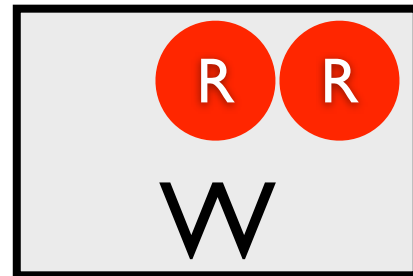
$$P(\text{win}) = ?$$

Marginal
Probability

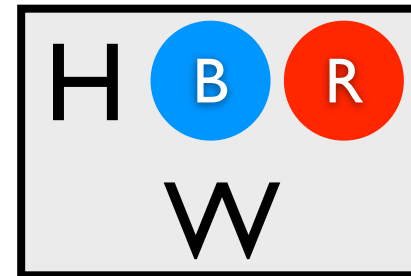
0.024



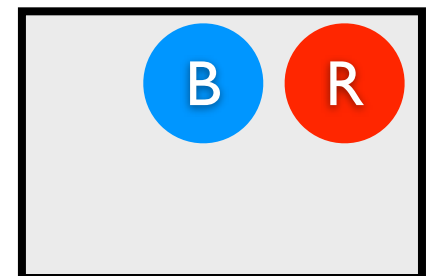
0.036



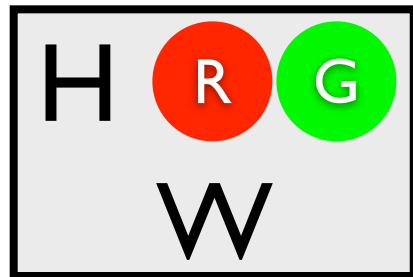
0.056



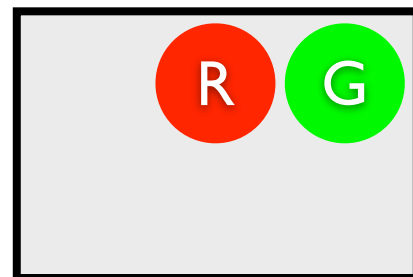
0.084



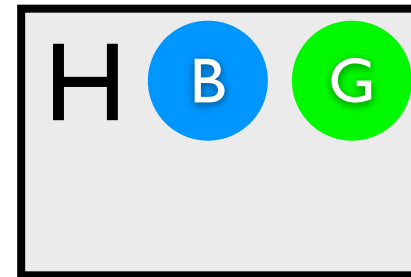
0.036



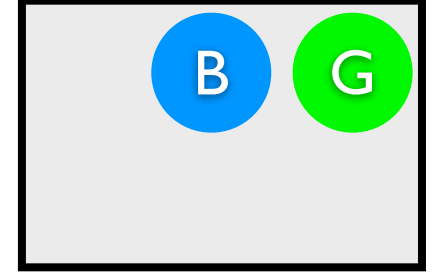
0.054



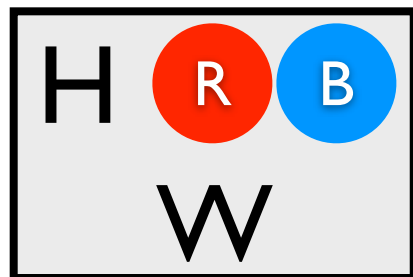
0.084



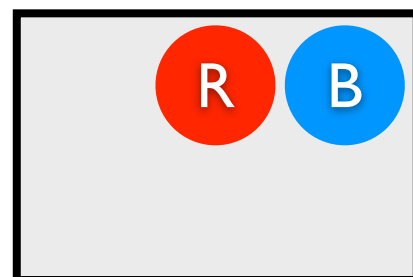
0.126



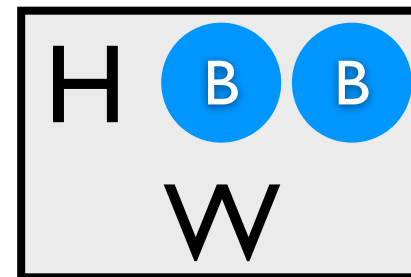
0.060



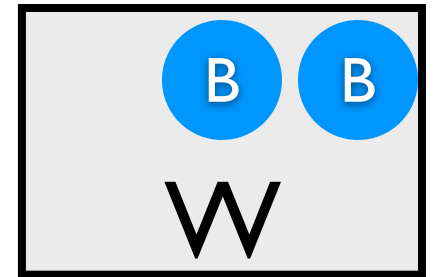
0.090



0.140



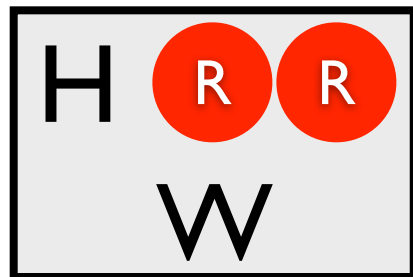
0.210



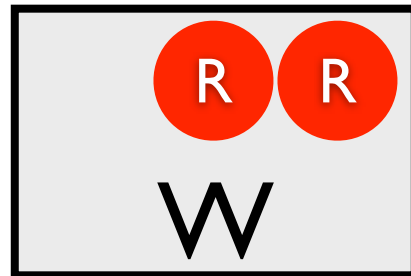
$$P(\text{win}) = \Sigma$$

Marginal
Probability

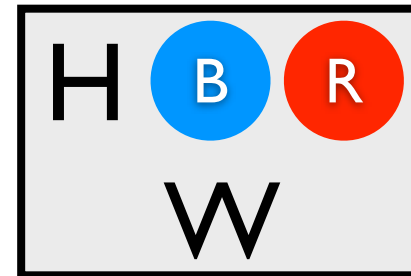
0.024



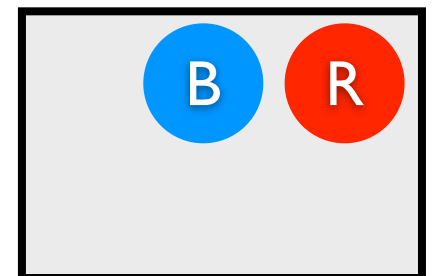
0.036



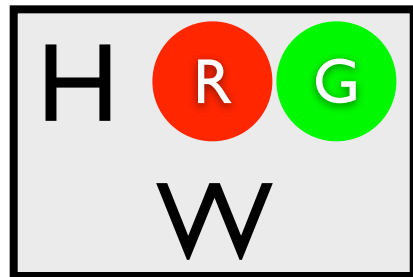
0.056



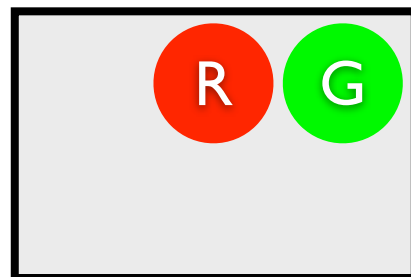
0.084



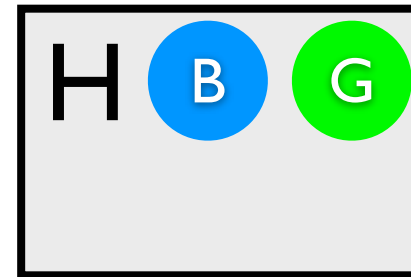
0.036



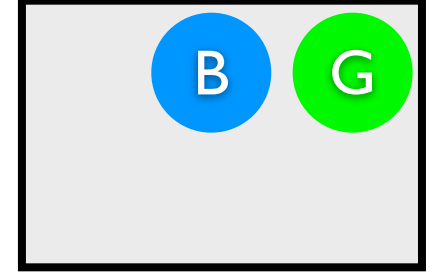
0.054



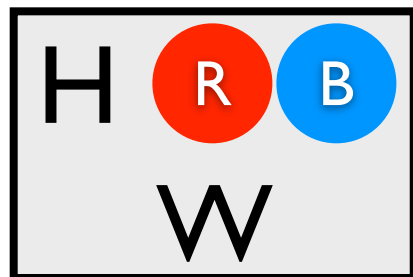
0.084



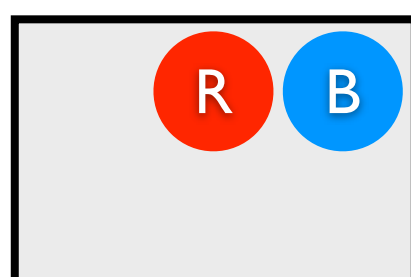
0.126



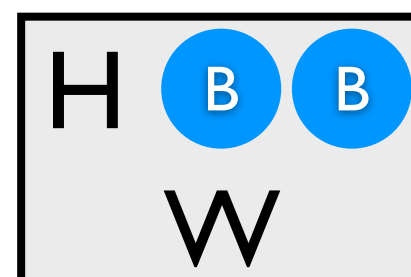
0.060



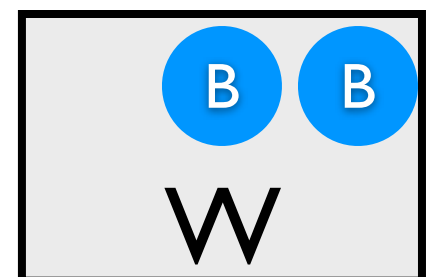
0.090



0.140



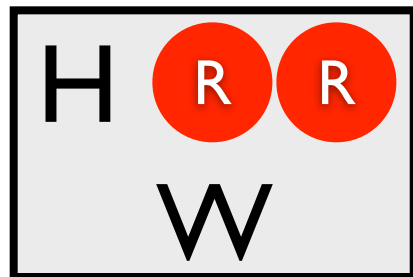
0.210



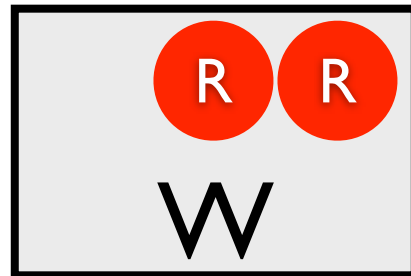
$$P(\text{win}) = \Sigma = 0.562$$

Marginal
Probability

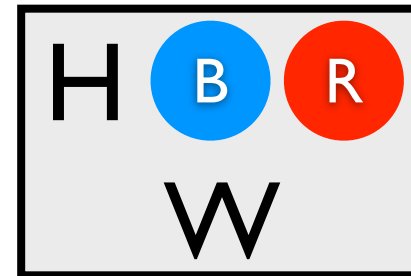
0.024



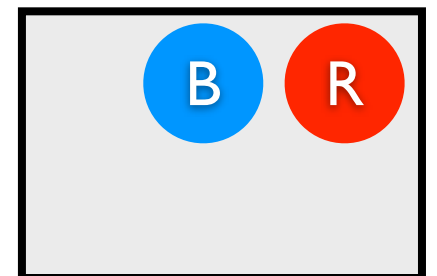
0.036



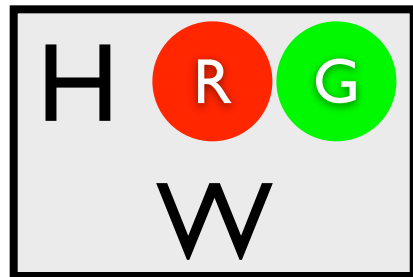
0.056



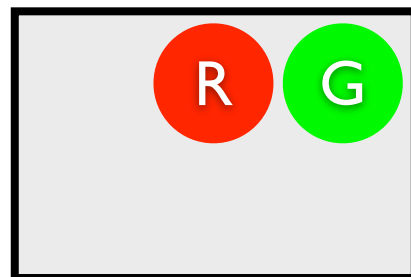
0.084



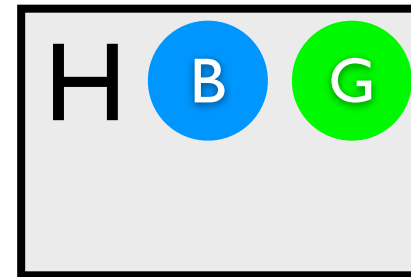
0.036



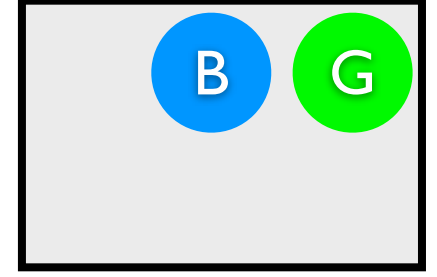
0.054



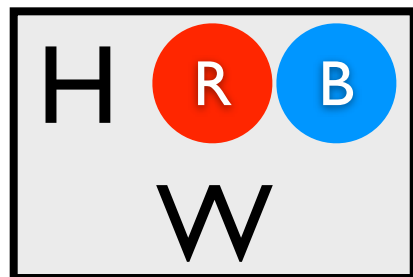
0.084



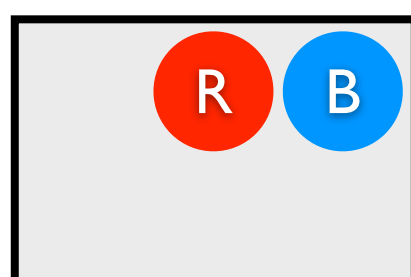
0.126



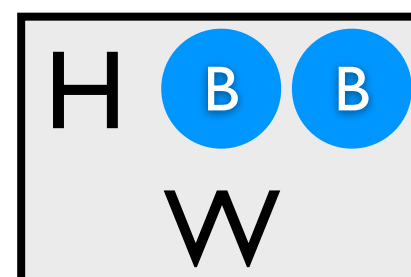
0.060



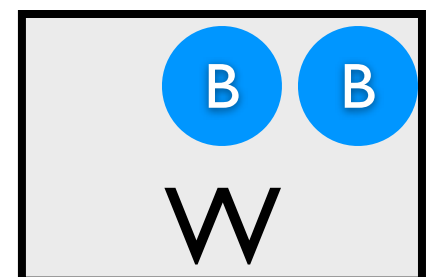
0.090



0.140



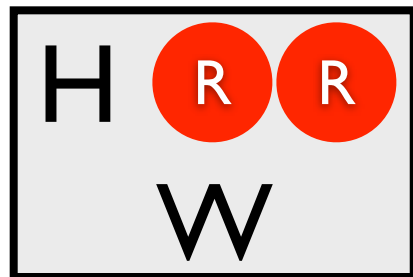
0.210



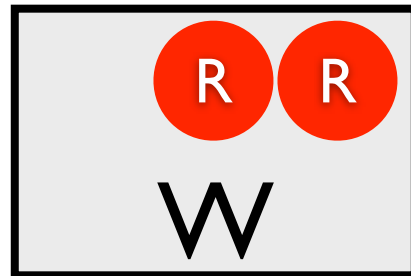
$$P(\text{win}|\text{col}(2,\text{green})) = ?$$

Conditional
Probability

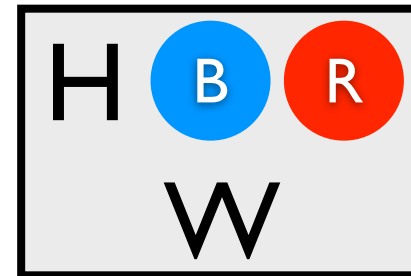
0.024



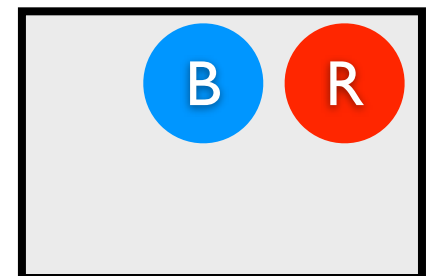
0.036



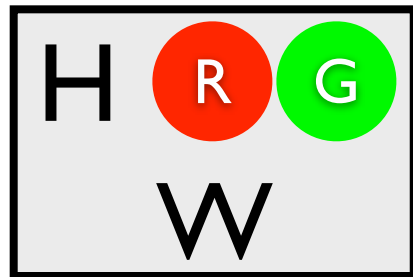
0.056



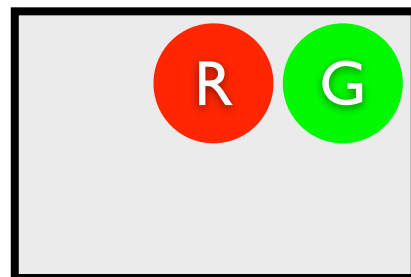
0.084



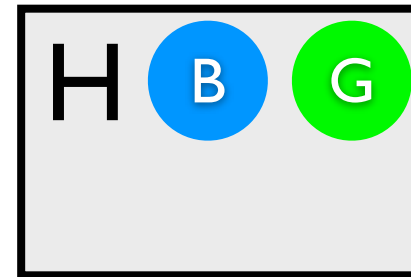
0.036



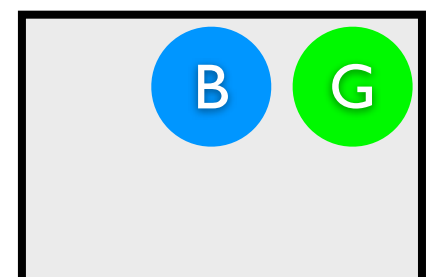
0.054



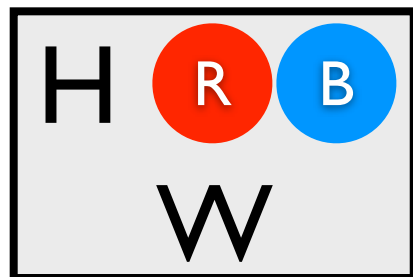
0.084



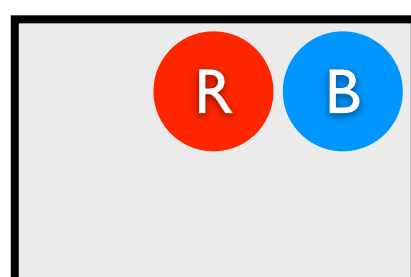
0.126



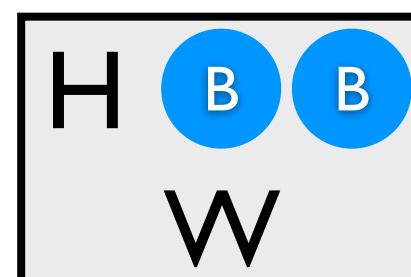
0.060



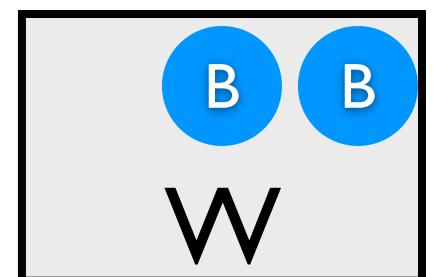
0.090



0.140



0.210

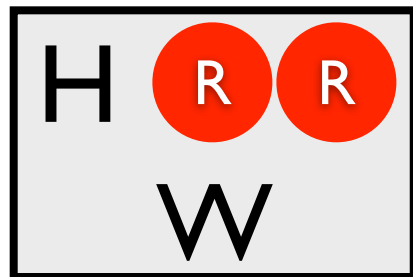


$$P(\text{win}|\text{col}(2,\text{green})) = \frac{\sum}{\Sigma}$$

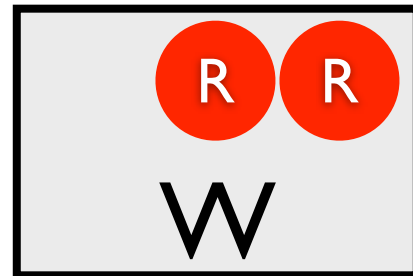
$$= \frac{P(\text{win} \wedge \text{col}(2,\text{green}))}{P(\text{col}(2,\text{green}))}$$

Conditional Probability

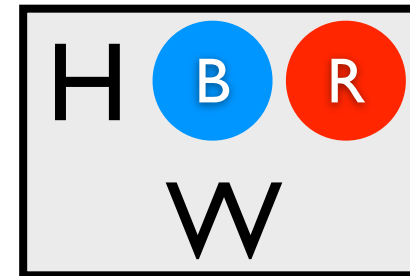
0.024



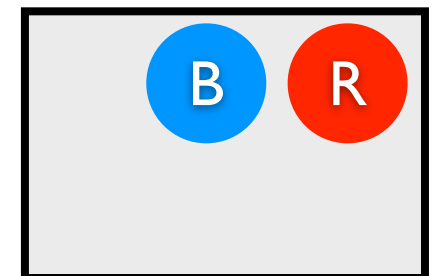
0.036



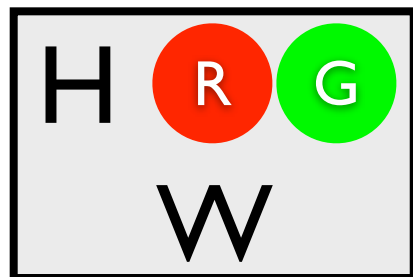
0.056



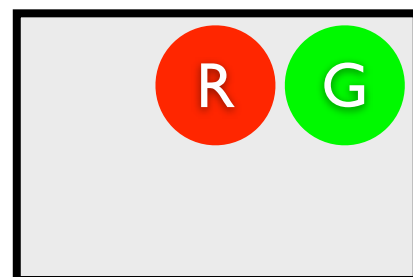
0.084



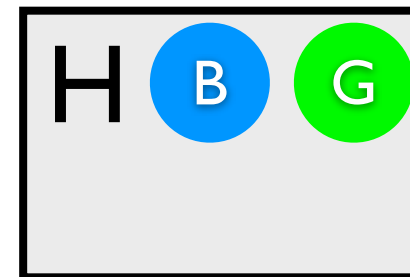
0.036



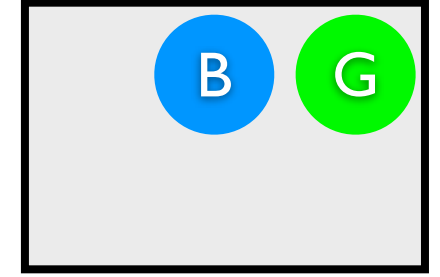
0.054



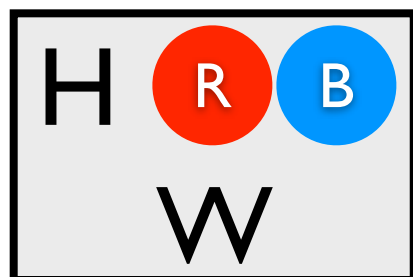
0.084



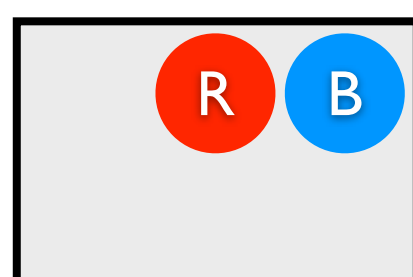
0.126



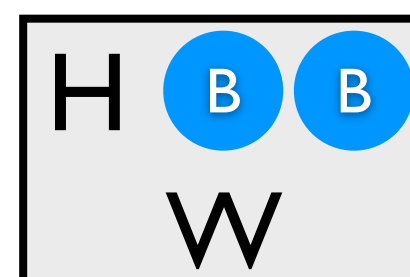
0.060



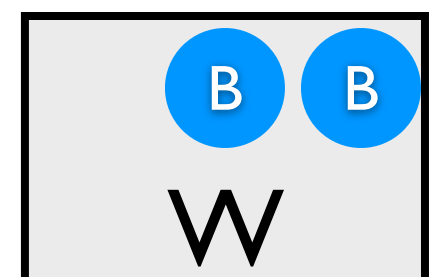
0.090



0.140



0.210

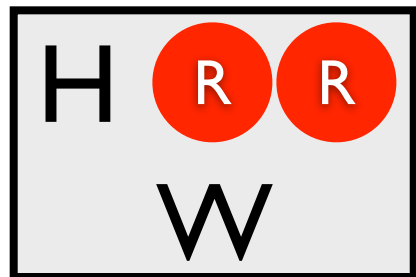


$$P(\text{win}|\text{col}(2,\text{green})) = \frac{\sum}{\Sigma}$$

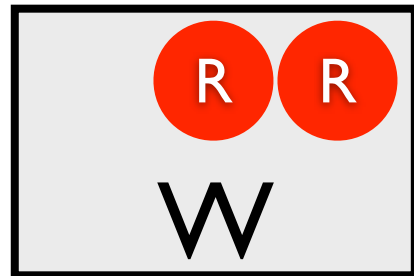
$$= \frac{P(\text{win} \wedge \text{col}(2,\text{green}))}{P(\text{col}(2,\text{green}))}$$

Conditional Probability

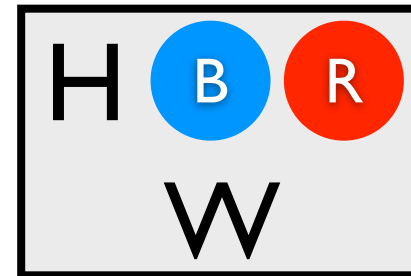
0.024



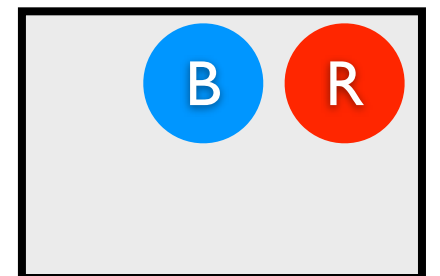
0.036



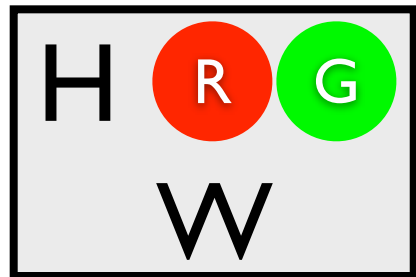
0.056



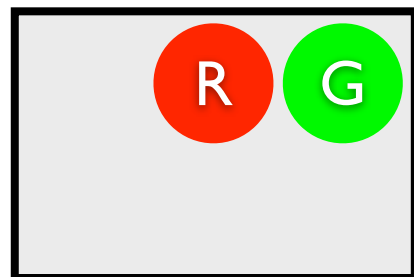
0.084



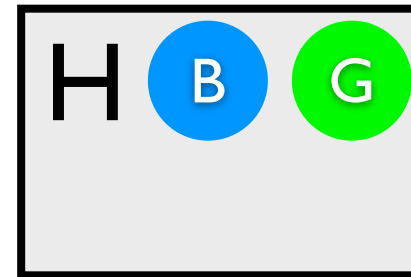
0.036



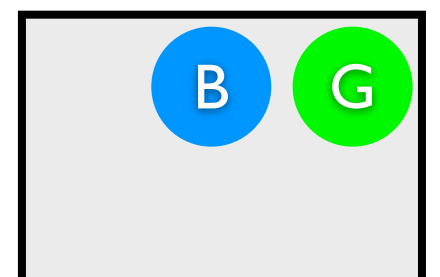
0.054



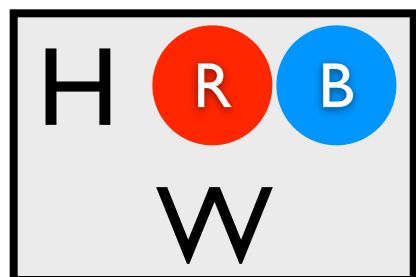
0.084



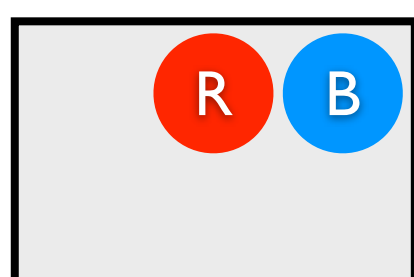
0.126



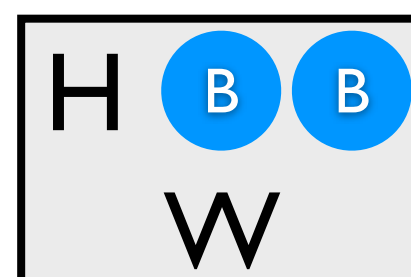
0.060



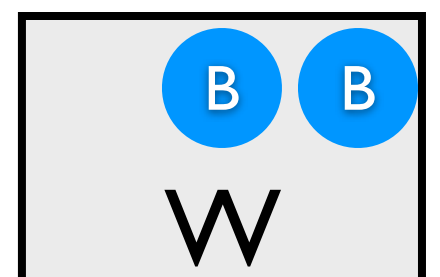
0.090



0.140



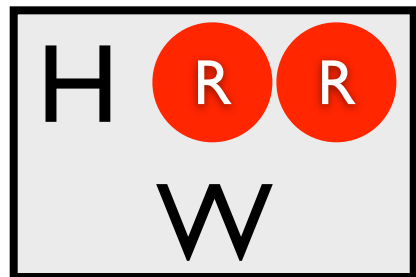
0.210



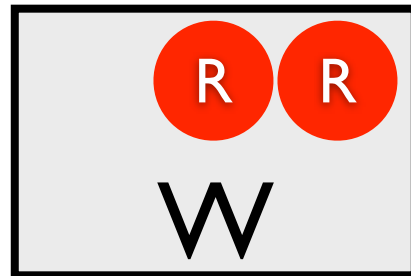
$$P(\text{win}|\text{col}(2,\text{green})) = \frac{\Sigma}{\Sigma} = 0.036/0.3 = 0.12$$

Conditional Probability

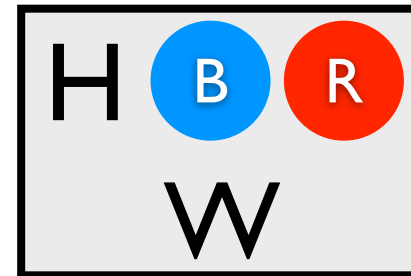
0.024



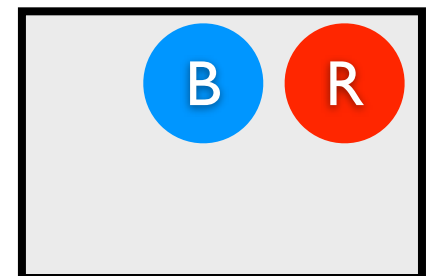
0.036



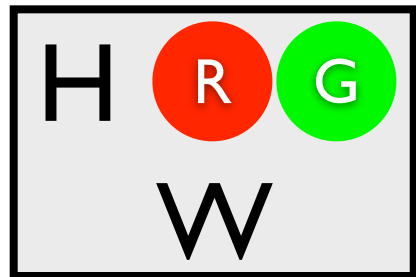
0.056



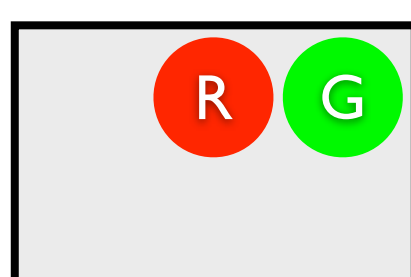
0.084



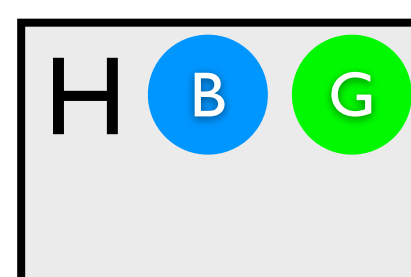
0.036



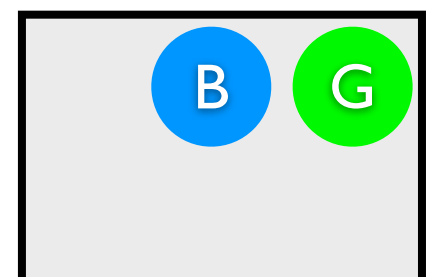
0.054



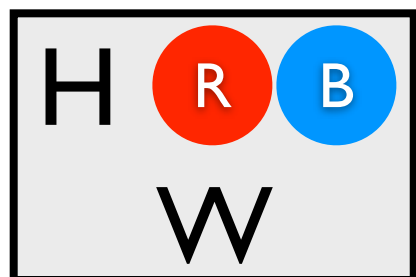
0.084



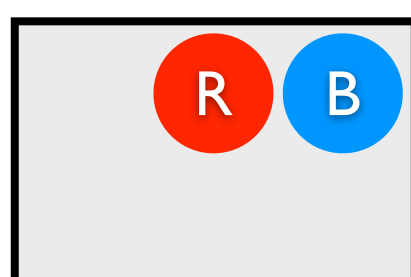
0.126



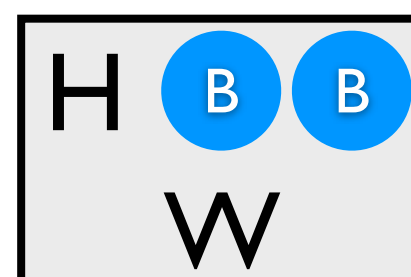
0.060



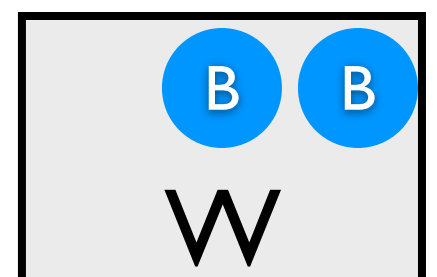
0.090



0.140



0.210

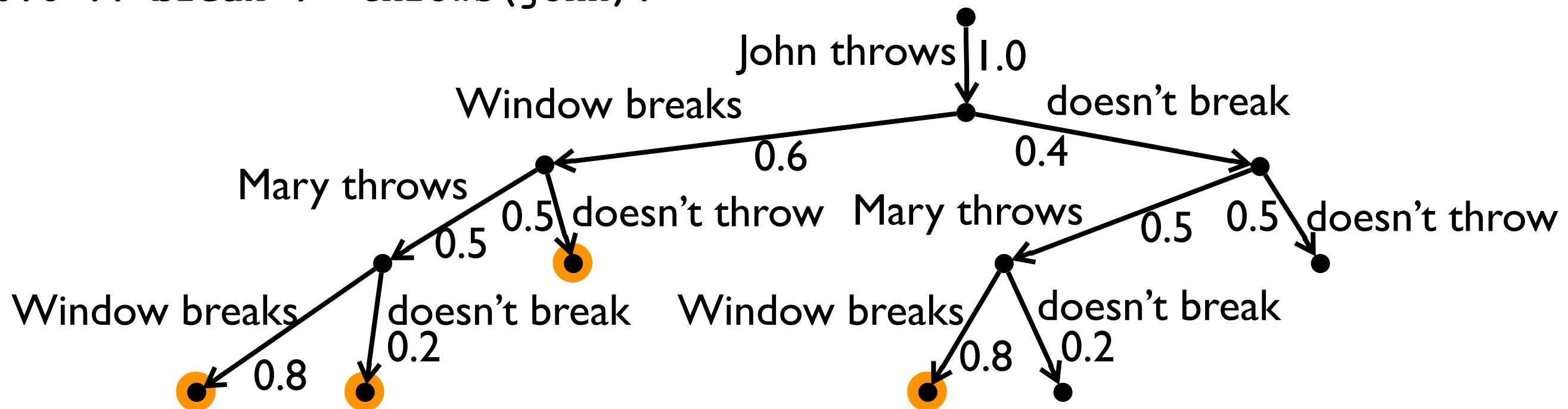


Alternative view: CP-Logic

```
throws(john) .
0.5 :: throws(mary) .
```

probabilistic causal laws

```
0.8 :: break :- throws(mary) .
0.6 :: break :- throws(john) .
```



$$P(\text{break}) = 0.6 \times 0.5 \times 0.8 + 0.6 \times 0.5 \times 0.2 + 0.6 \times 0.5 + 0.4 \times 0.5 \times 0.8$$

Distributional Clauses (DC)

Closely related to BLOG [Russell et al.]

- Discrete- and continuous-valued random variables

Distributional Clauses (DC)

Closely related to BLOG [Russell et al.]

- Discrete- and continuous-valued random variables

random variable with Gaussian distribution

```
length(Obj) ~ gaussian(6.0, 0.45) :- type(Obj, glass) .
```



Distributional Clauses (DC)

Closely related to BLOG [Russell et al.]

- Discrete- and continuous-valued random variables

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).  
stackable(ObjBot,ObjTop) :-
```

```
     $\simeq \text{length}(\text{ObjBot}) \geq \simeq \text{length}(\text{ObjTop})$  ,  
     $\simeq \text{width}(\text{ObjBot}) \geq \simeq \text{width}(\text{ObjTop})$  .
```

comparing values of
random variables



Distributional Clauses (DC)

Closely related to BLOG [Russell et al.]

- Discrete- and continuous-valued random variables

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
```

```
stackable(Obj1,Obj2) :-
```

```
    ≈length(Obj1) ≥ ≈length(Obj2),
```

```
    ≈width(Obj1) ≥ ≈width(Obj2).
```

```
ontype(Obj,plate) ~ finite([0 : glass, 0.0024 : cup,  
                             0 : pitcher, 0.8676 : plate,  
                             0.0284 : bowl, 0 : serving,  
                             0.1016 : none])
```

```
:- obj(Obj), on(Obj,O2), type(O2,plate).
```

**random variable with
discrete distribution**



Distributional Clauses (DC)

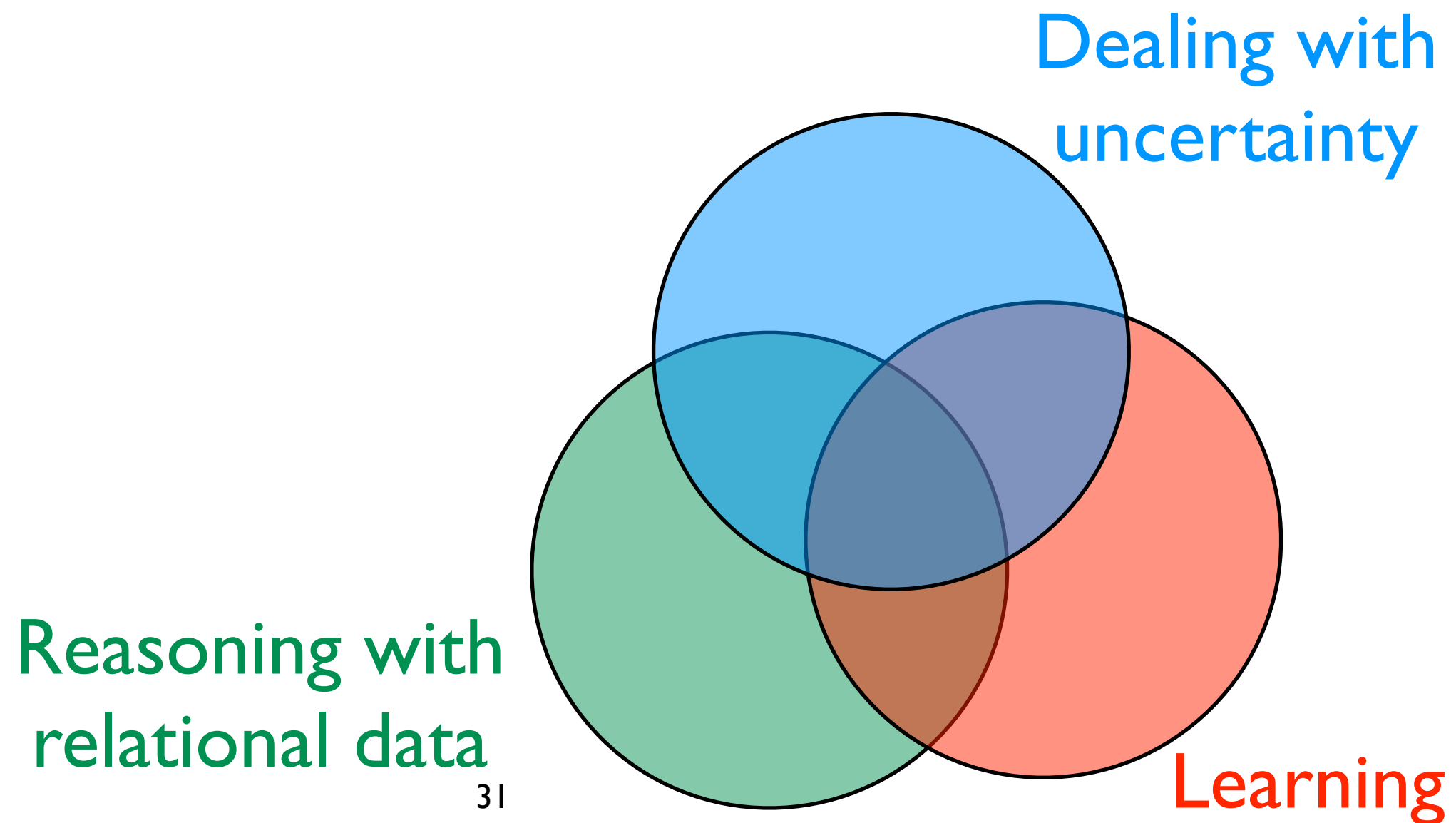
Closely related to BLOG [Russell et al.]

- Discrete- and continuous-valued random variables

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
stackable(Obj1,Obj2) :-
    ≈length(Obj1) ≥ ≈length(Obj2),
    ≈width(Obj1) ≥ ≈width(Obj2).
ontype(Obj,plate) ~ finite([0 : glass, 0.0024 : cup,
                           0 : pitcher, 0.8676 : plate,
                           0.0284 : bowl, 0 : serving,
                           0.1016 : none])
:- obj(Obj), on(Obj,Obj2), type(Obj2,plate).
```



Probabilistic Databases



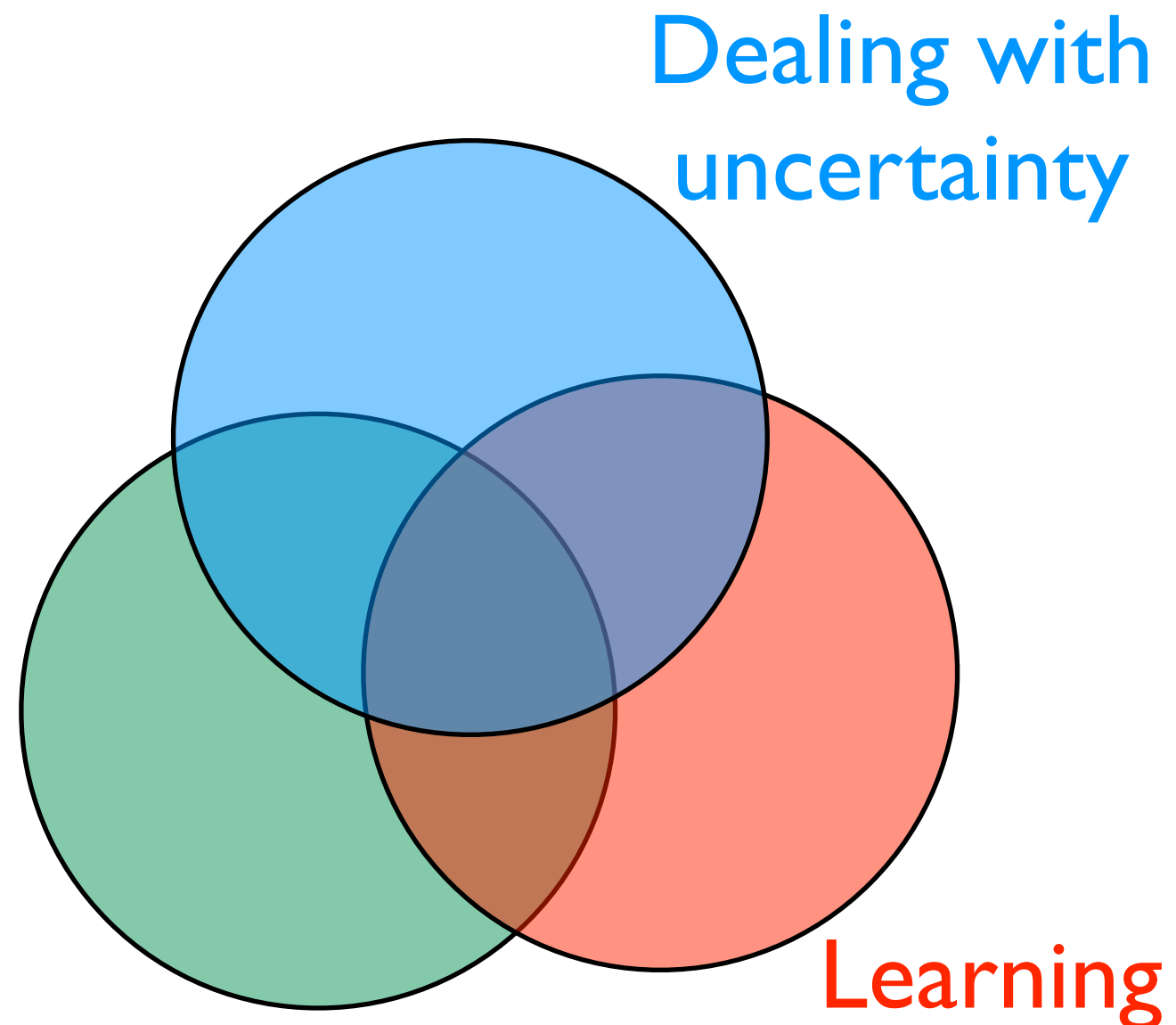
Probabilistic Databases

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

bornIn	
person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn	
city	country
london	uk
york	uk
paris	usa

relational
database



Probabilistic Databases

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

one world

bornIn

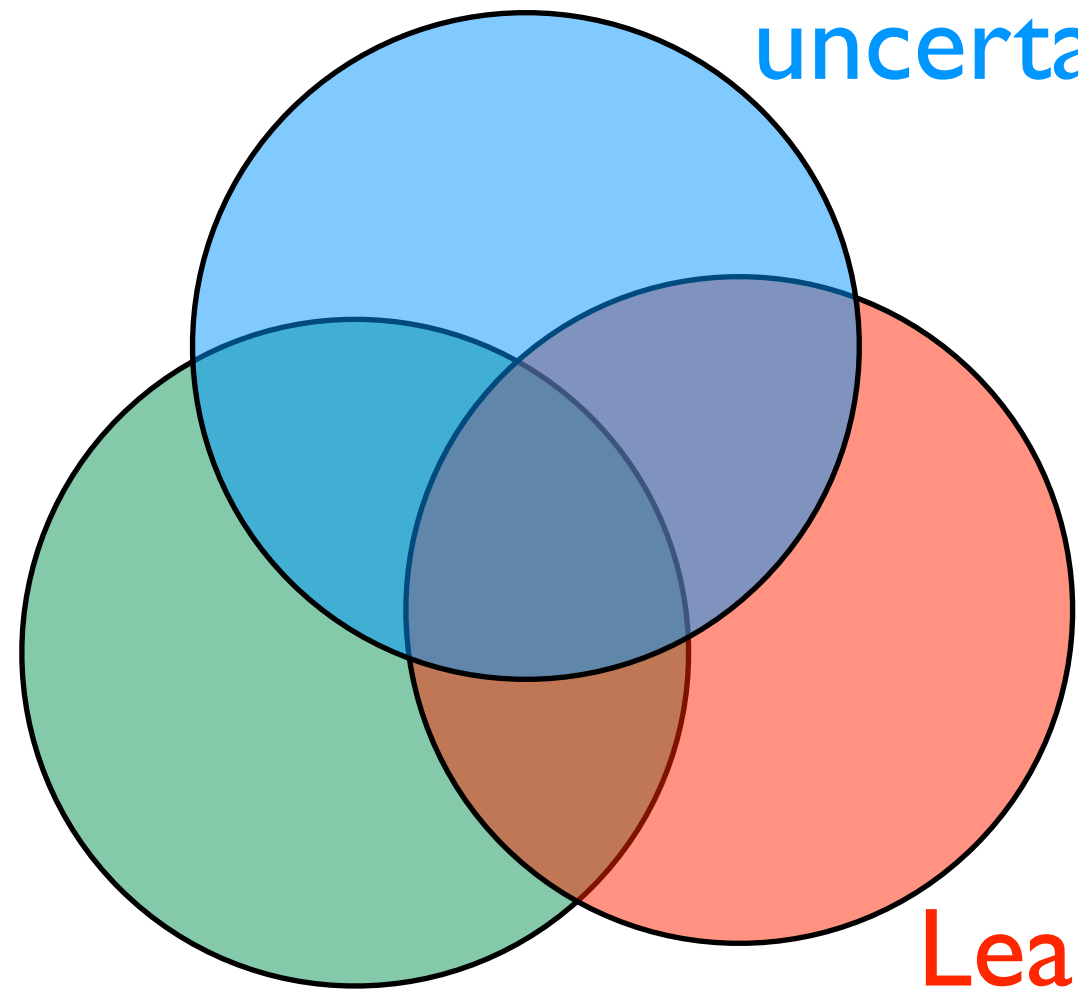
person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational
database

Dealing with
uncertainty



Learning

Probabilistic Databases

bornIn		
person	city	P
ann	london	0,87
bob	york	0,95
eve	new york	0,9
tom	paris	0,56

cityIn		
city	country	P
london	uk	0,99
york	uk	0,75
paris	usa	0,4

tuples as random
variables

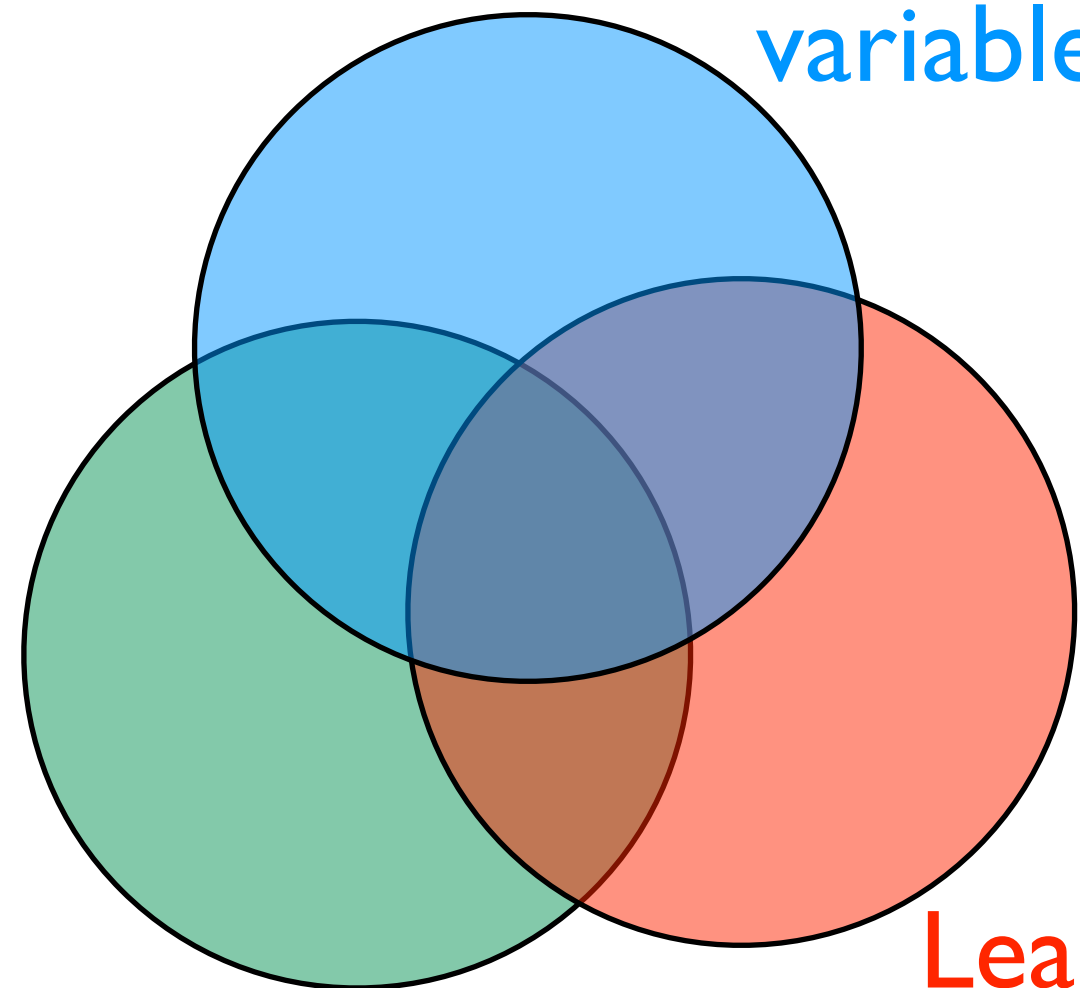
```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

one world

bornIn	
person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn	
city	country
london	uk
york	uk
paris	usa

relational
database



Learning

Probabilistic Databases

several possible worlds

bornIn		
person	city	P
ann	london	0,87
bob	york	0,95
eve	new york	0,9
tom	paris	0,56

cityIn		
city	country	P
london	uk	0,99
york	uk	0,75
paris	usa	0,4

tuples as random

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

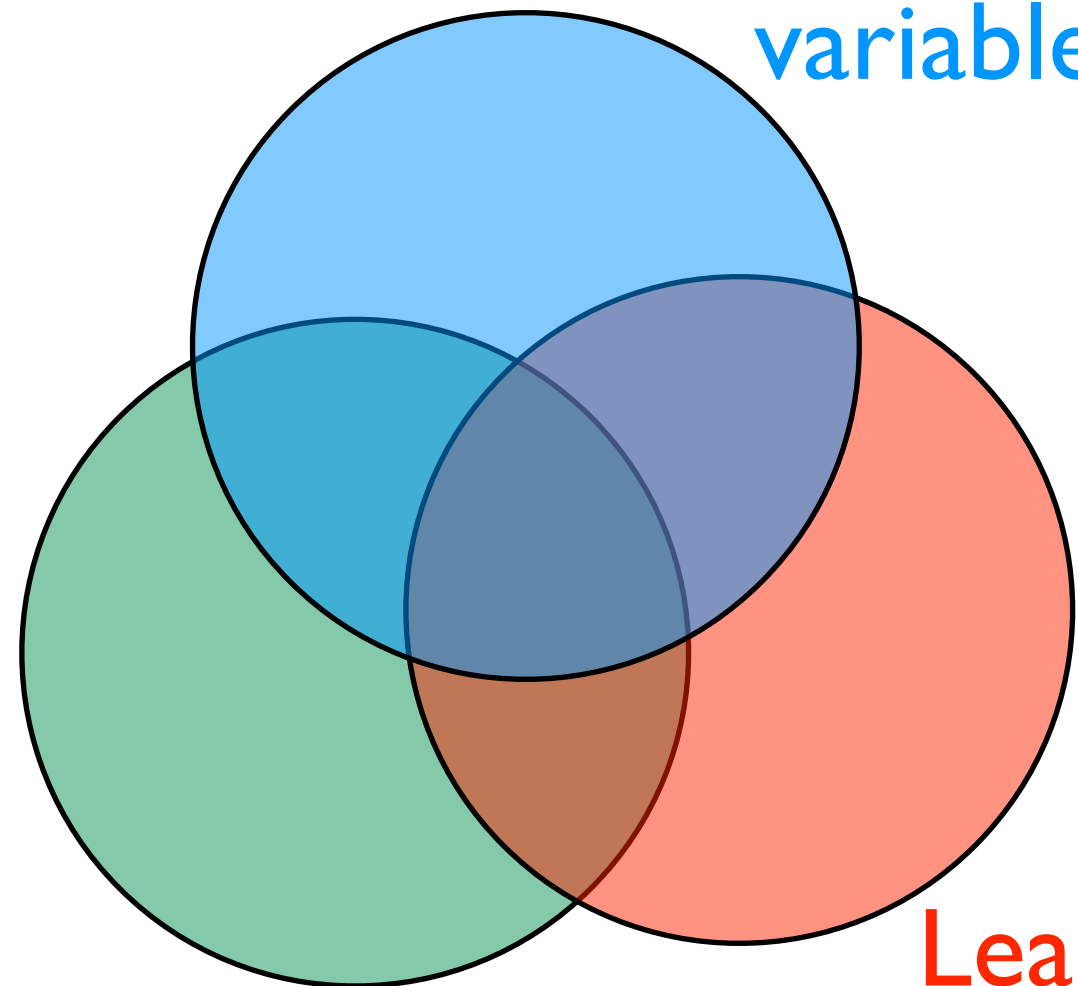
one world

bornIn	
person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn	
city	country
london	uk
york	uk
paris	usa

relational
database

variables



Learning

Probabilistic Databases

several possible worlds

bornIn		
person	city	P
ann	london	0,87
bob	york	0,95
		0,9
		0,56

cityIn		
city	country	P
london	uk	0,99
york	uk	0,75
paris	usa	0,4

probabilistic tables + database queries
→ distribution over possible worlds

tuples as random
variables

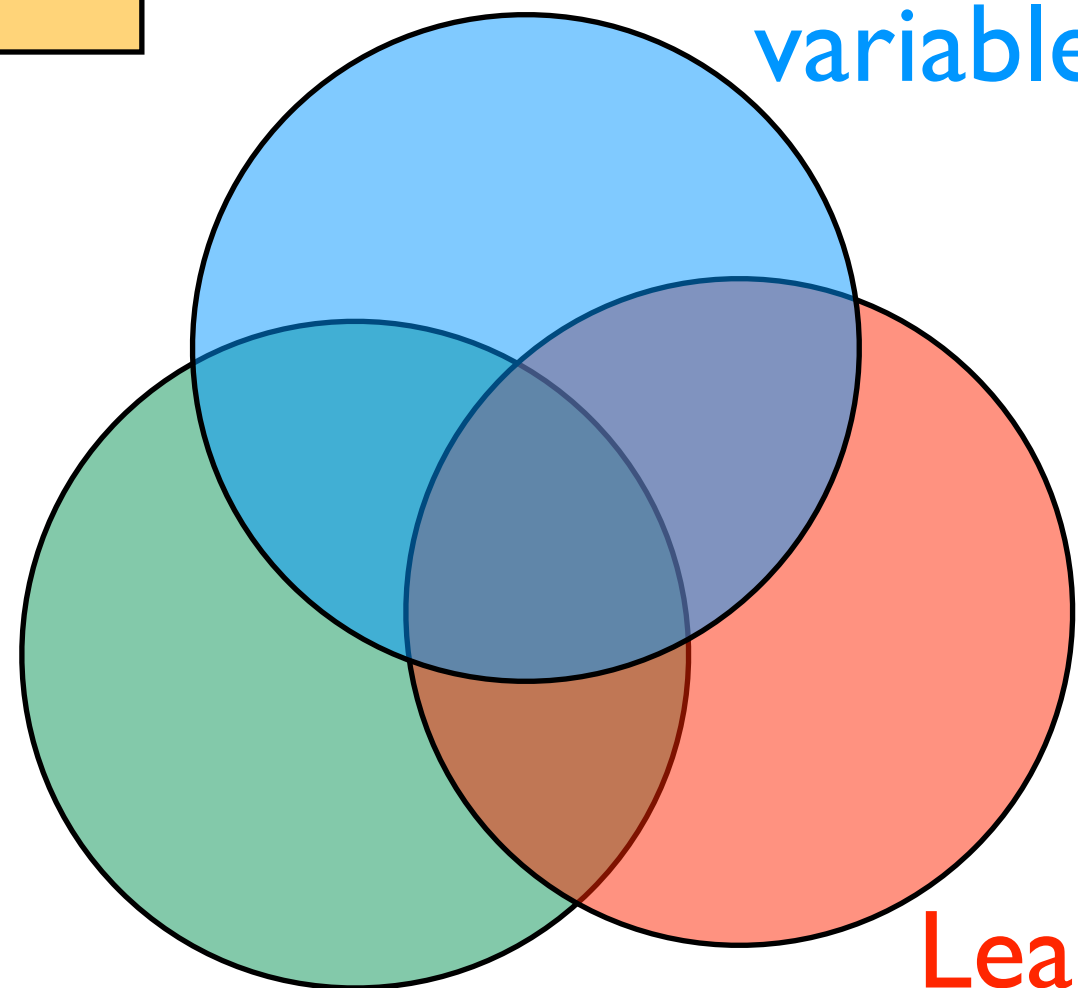
```
select
from bornIn x, cityIn y
where x.city=y.city
```

one world

bornIn	
person	city
ann	london
bob	york
eve	new york
tom	paris


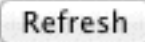
cityIn	
city	country
london	uk
york	uk
paris	usa





















relational
database



Learning

Example: Information Extraction

Recently-Learned Facts  

instance	iteration	date learned	confidence
<u>kelly andrews</u> is a <u>female</u>	826	29-mar-2014	98.7  
<u>investment next year</u> is an <u>economic sector</u>	829	10-apr-2014	95.3  
<u>shibenik</u> is a <u>geopolitical entity</u> that is an organization	829	10-apr-2014	97.2  
<u>quality web design work</u> is a <u>character trait</u>	826	29-mar-2014	91.0  
<u>mercedes benz cls by carlsson</u> is an <u>automobile manufacturer</u>	829	10-apr-2014	95.2  
<u>social work</u> is an academic program <u>at the university rutgers university</u>	827	02-apr-2014	93.8  
<u>dante wrote</u> the book <u>the divine comedy</u>	826	29-mar-2014	93.8  
<u>willie aames</u> was <u>born in</u> the city <u>los angeles</u>	831	16-apr-2014	100.0  
<u>kitt peak</u> is a mountain <u>in the state or province arizona</u>	831	16-apr-2014	96.9  
<u>greenwich</u> is a park <u>in the city london</u>	831	16-apr-2014	100.0  

↑
instances for many
different relations

↑
degree of certainty

Distribution Semantics

- **probabilistic choices** + their **consequences**
- probability distribution over **possible worlds**
- how to efficiently answer **questions?**
 - most probable world (MPE inference)
 - probability of query (computing marginals)
 - probability of query given evidence

Summary: ProbLog Syntax

- input database: ground facts

```
person(bob) .
```

- probabilistic facts

```
0.5::stress(bob) .
```

- annotated disjunctions

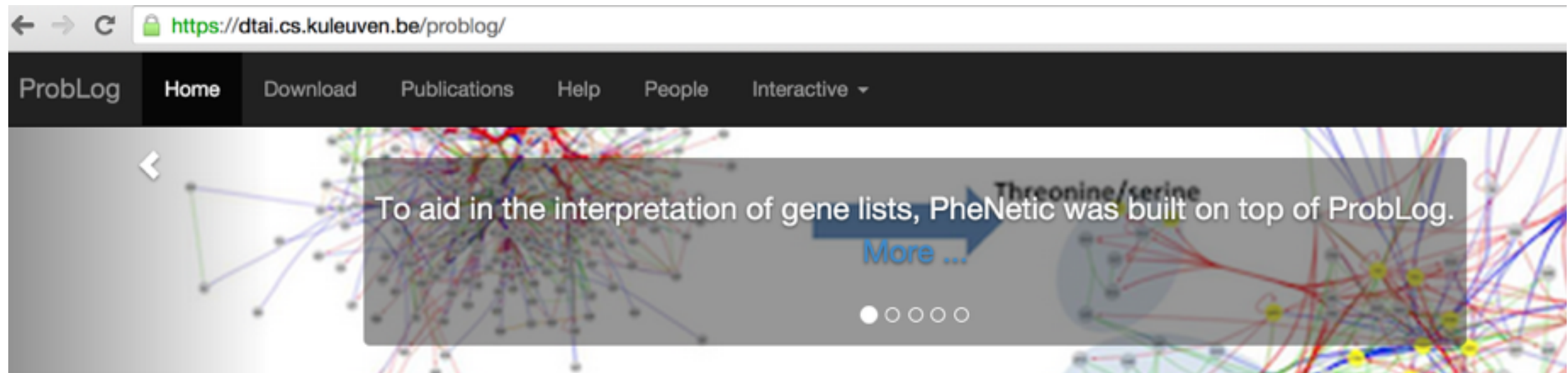
```
0.5::stress(X) :- person(X) .  
0.4::a(X) ; 0.3::b(X) ; 0.2::c(X) ; 0.1::d(X) :- q(X) .  
0.5::weather(sun,0) ; 0.5::weather(rain,0) .
```

- flexible probabilities

```
P::pack(Item) :- weight(Item,W), P is 1.0/W.
```

- Prolog clauses

```
smokes(X) :- influences(Y,X), smokes(Y) .  
excess([I|R],Limit) :- \+pack(I), excess(R,Limit) .
```



Introduction.

Probabilistic logic programs are logic programs in which some of the facts are annotated with probabilities.

ProbLog is a tool that allows you to intuitively build programs that do not only encode **complex interactions** between a large sets of **heterogenous components** but also **uncertainties** that are present in real-life situations.

The engine tackles several tasks such as computing the marginals given evidence and learning from (partial) interpretations. ProbLog is a suite of efficient algorithms for these tasks. It is based on a conversion of the program and the queries and evidence to a weighted Boolean formula. This allows us to reduce the inference tasks to well-solved model counting, which can be solved using state-of-the-art methods known from the graphical model and knowledge compilation literature.

The Language. Probabilistic Logic Programming.

ProbLog makes it easy to express complex, probabilistic models.

```
0.3::stress(X) :- person(X).
0.2::influences(X,Y) :- person(X), person(Y).

smokes(X) :- stress(X).
smokes(X) :- friend(X,Y), influences(Y,X), smokes(Y).
```

Some Probabilistic Programming Languages outside LP

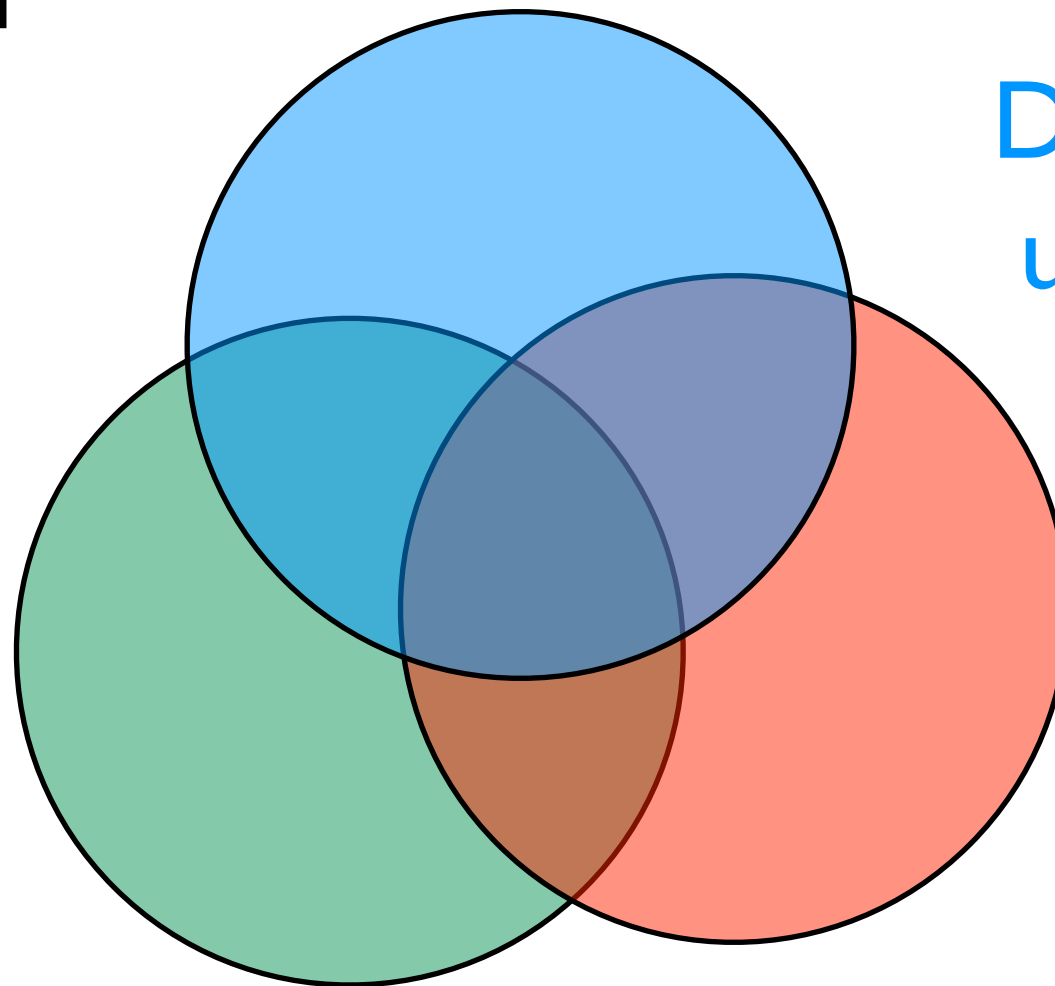
- IBAL [Pfeffer 01]
- Figaro [Pfeffer 09]
- Church [Goodman et al 08]
- BLOG [Milch et al 05]
- Venture [Mansingha et al.]
- Anglican and Probabilistic-C [Wood et al].
- and many more appearing recently

Church

probabilistic functional
programming

[Goodman et al, UAI 08]

Reasoning with
relational data



Dealing with
uncertainty

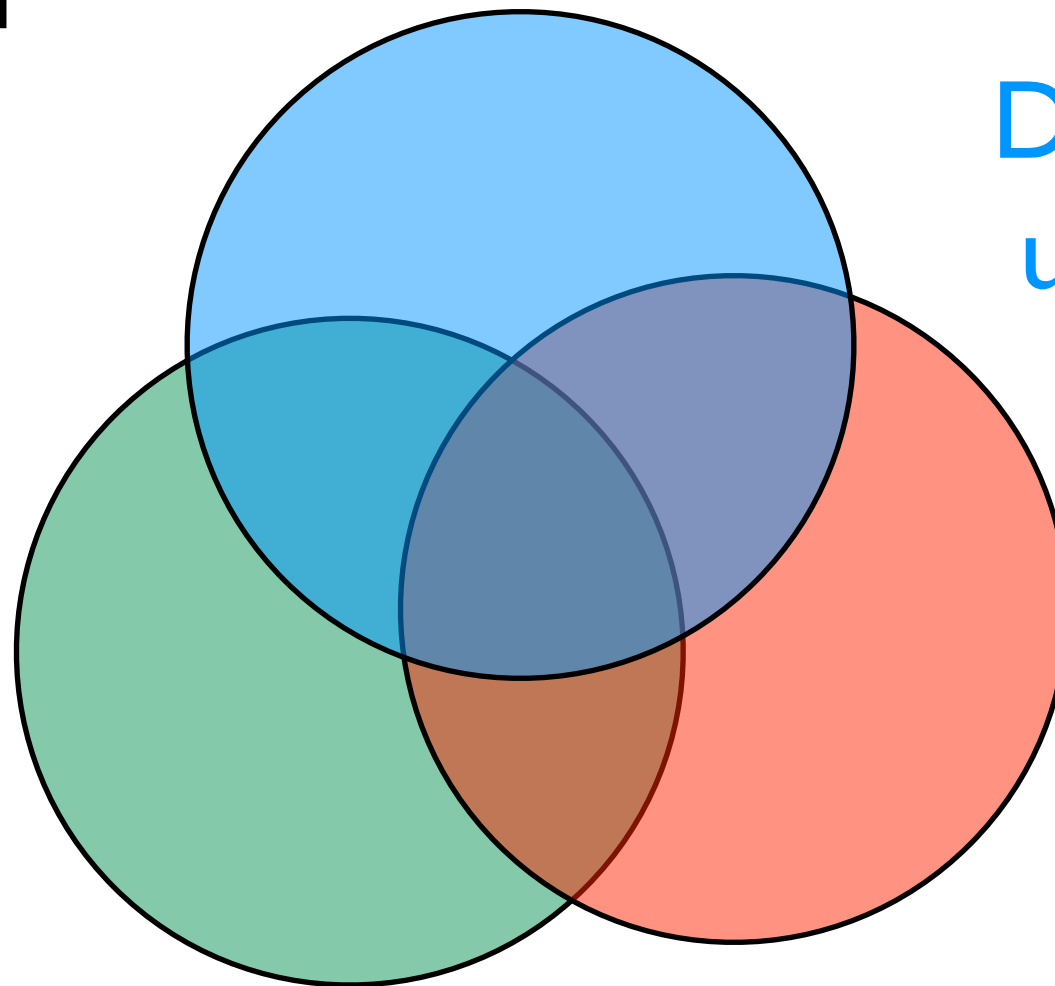
Learning

Church

probabilistic functional
programming

[Goodman et al, UAI 08]

functional
programming



Dealing with
uncertainty

Learning

```
(define plus5 (lambda (x) (+ x 5)))
```

```
(map plus5 '(1 2 3))
```

Church

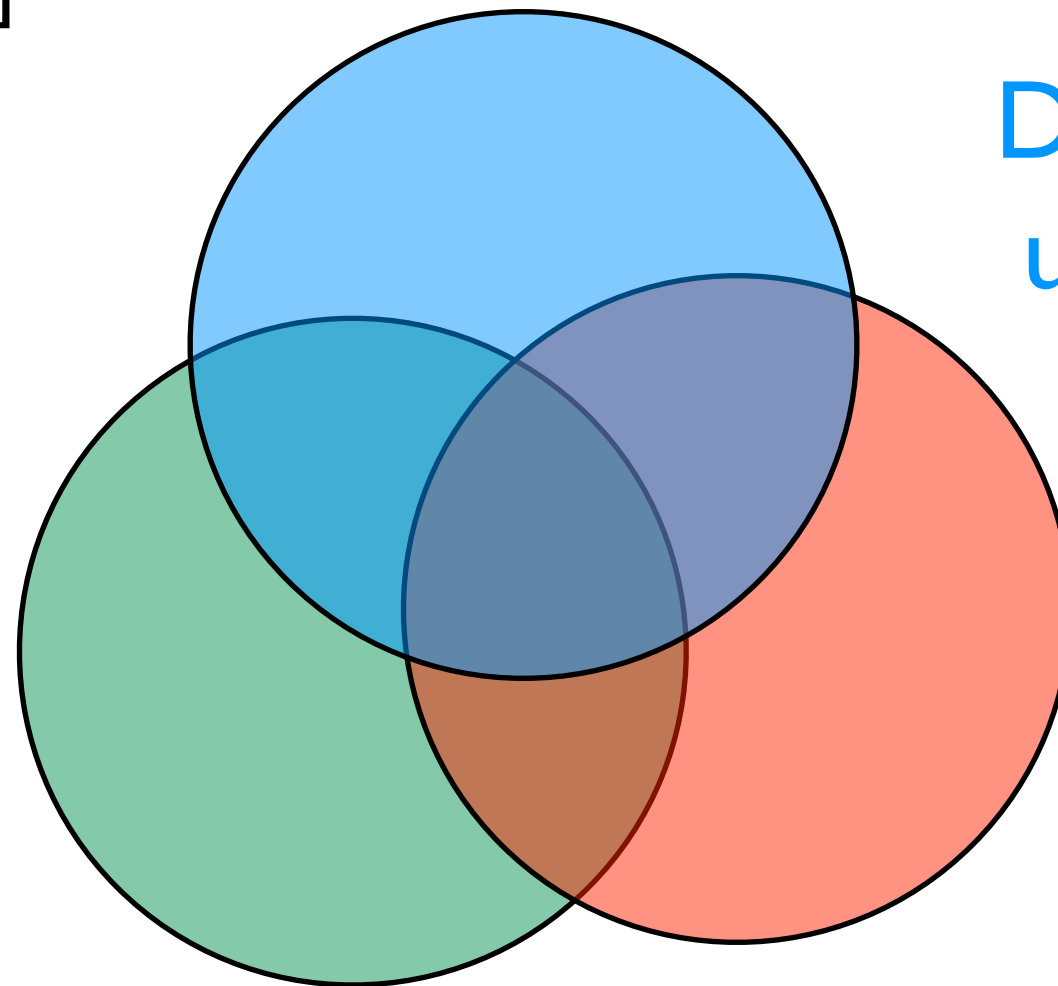
probabilistic functional
programming

[Goodman et al, UAI 08]

functional
programming

one execution

```
(define plus5 (lambda (x) (+ x 5)))  
  
(map plus5 '(1 2 3))
```



Dealing with
uncertainty

Learning

Church

probabilistic functional

programming

[Goodman et al, UAI 08]

```
(define randplus5  
  (lambda (x) (if (flip 0.6)  
                  (+ x 5)  
                  x)))
```

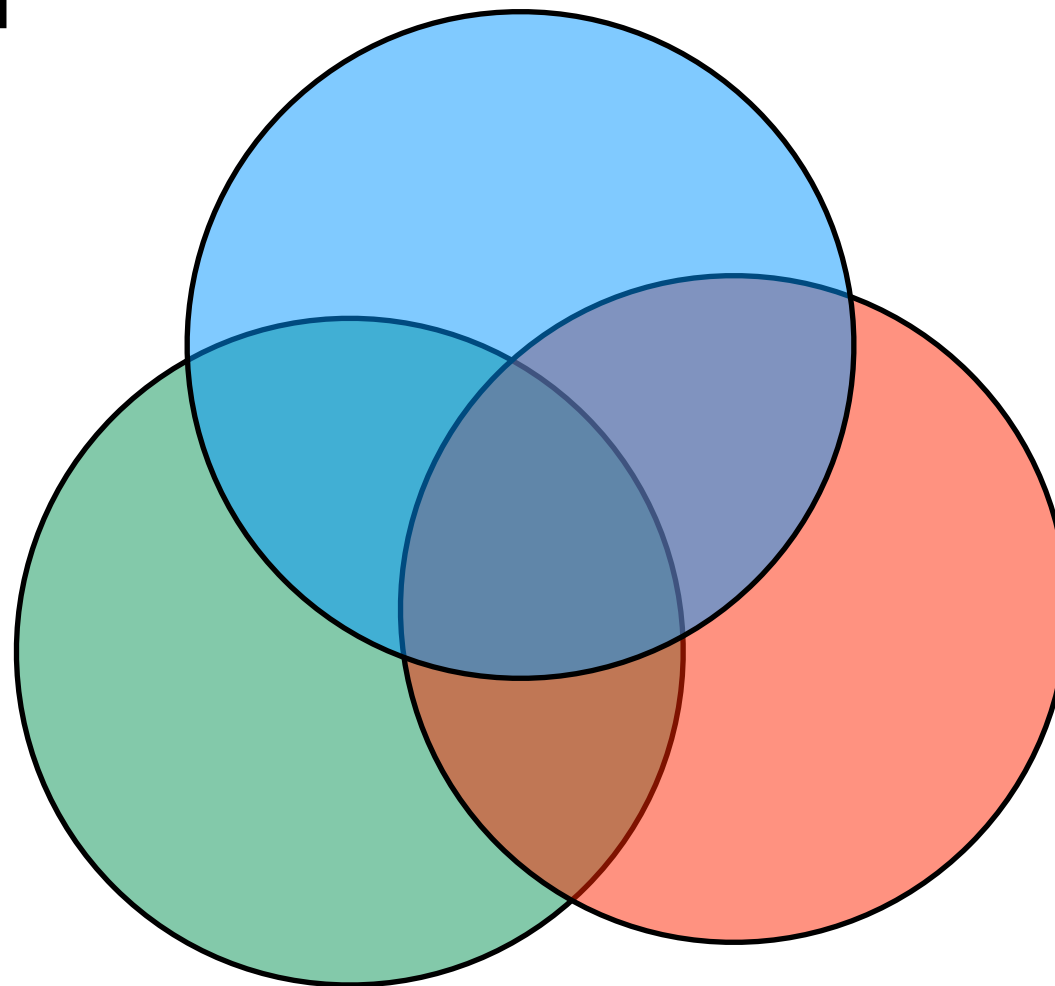
```
(map randplus5 '(1 2 3))
```

random
primitives

functional
programming

one execution

```
(define plus5 (lambda (x) (+ x 5)))  
  
(map plus5 '(1 2 3))
```



Learning

Church

probabilistic functional
programming

[Goodman et al, UAI 08]

**several
possible
executions**

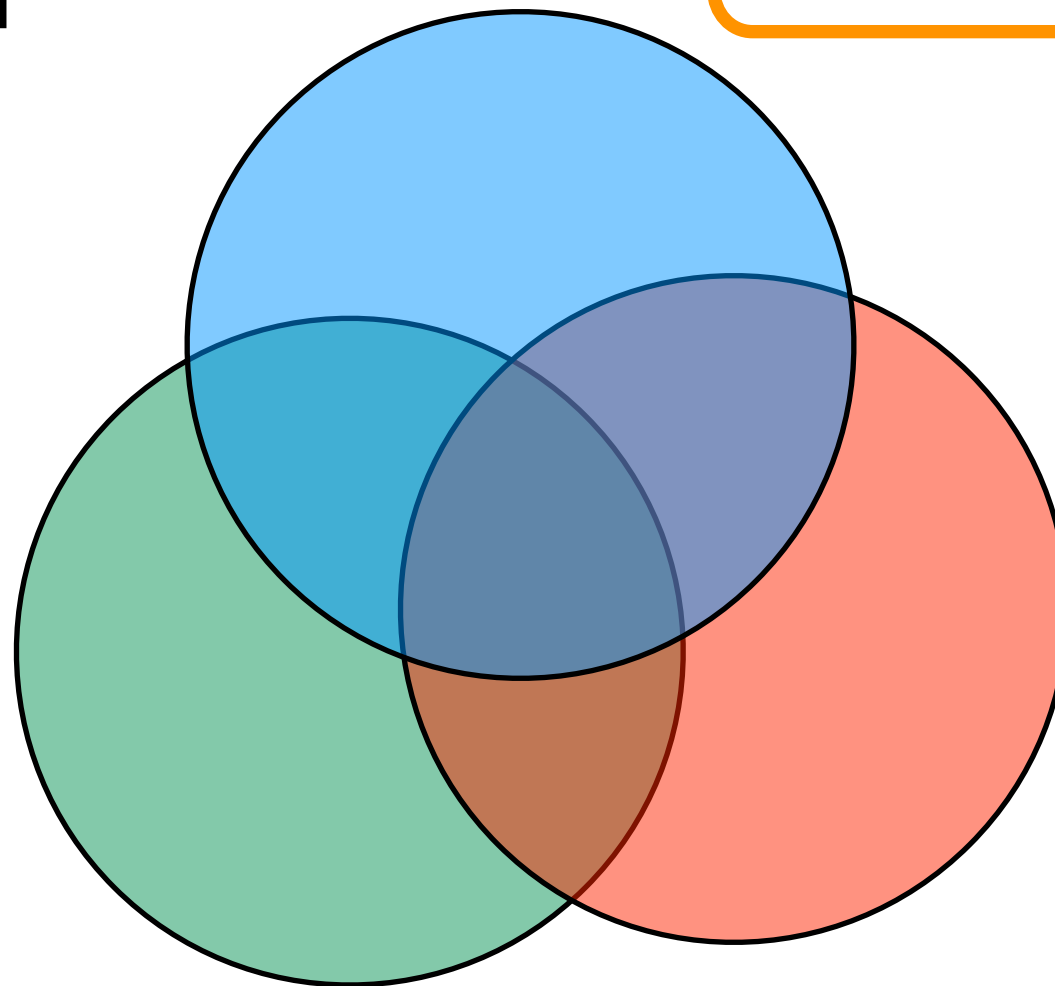
```
(define randplus5  
  (lambda (x) (if (flip 0.6)  
                  (+ x 5)  
                  x)))  
  
(map randplus5 '(1 2 3))
```

random
primitives

functional
programming

one execution

```
(define plus5 (lambda (x) (+ x 5)))  
  
(map plus5 '(1 2 3))
```



Learning

Church

probabilistic functional
programming

[Goodman et al, UAI 08]

**several
possible
executions**

```
(define randplus5  
  (lambda (x) (if (flip 0.6)  
                  (+ x 5)  
                  x)))  
  
(map randplus5 '(1 2 3))
```

probabilistic primitives + functional program
→ distribution over possible executions

random
primitives

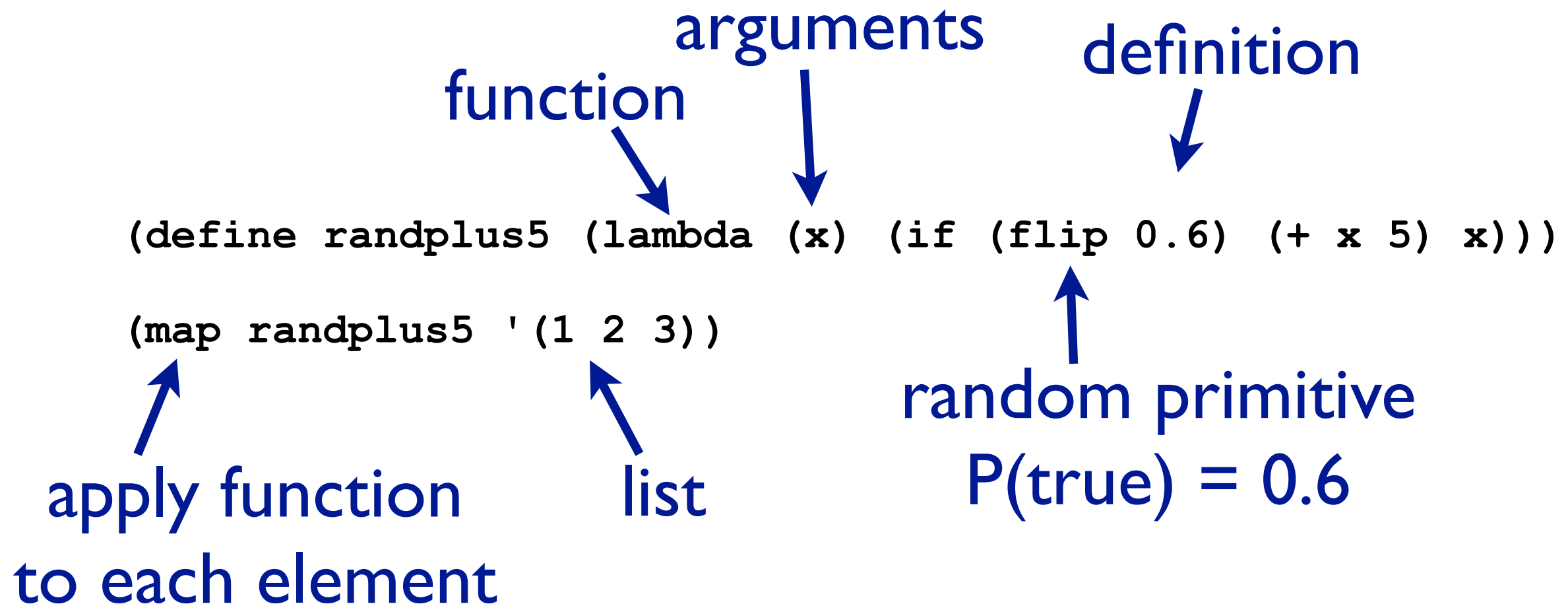
functional
programming

Learning

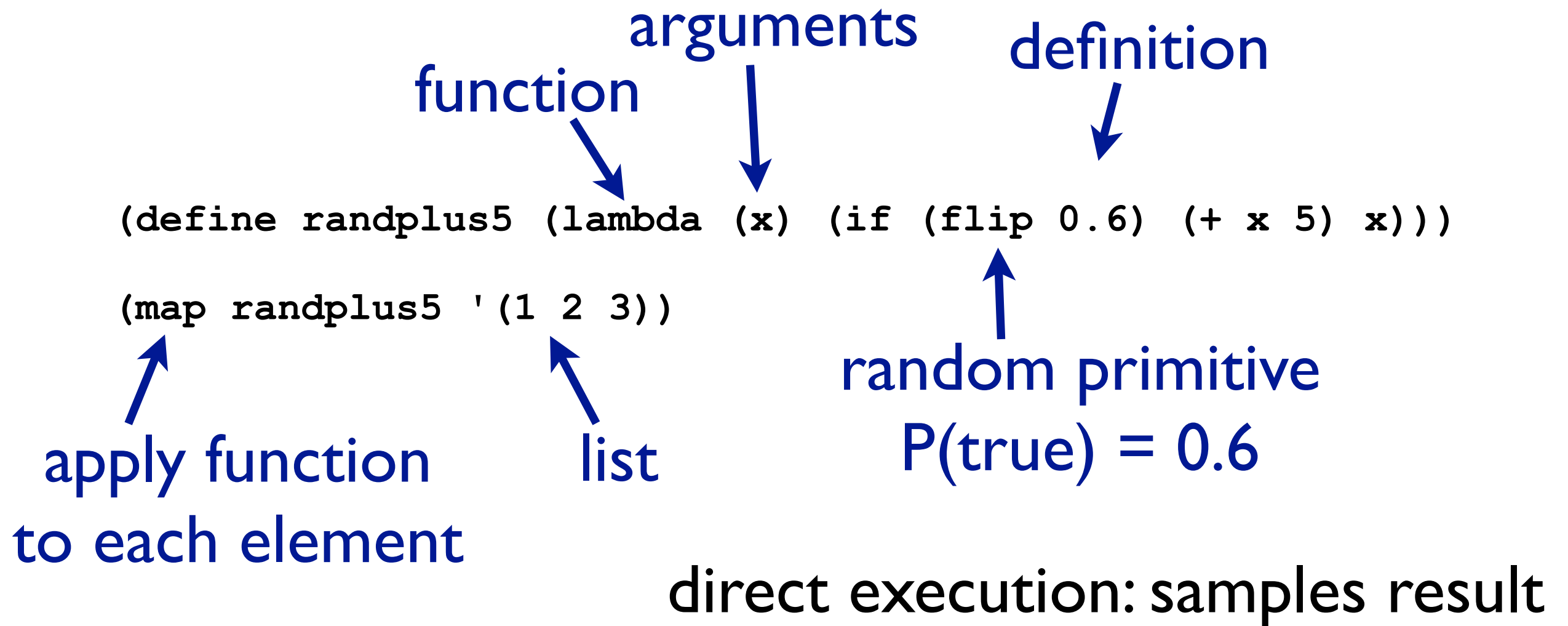
one execution

```
(define plus5 (lambda (x) (+ x 5)))  
  
(map plus5 '(1 2 3))
```

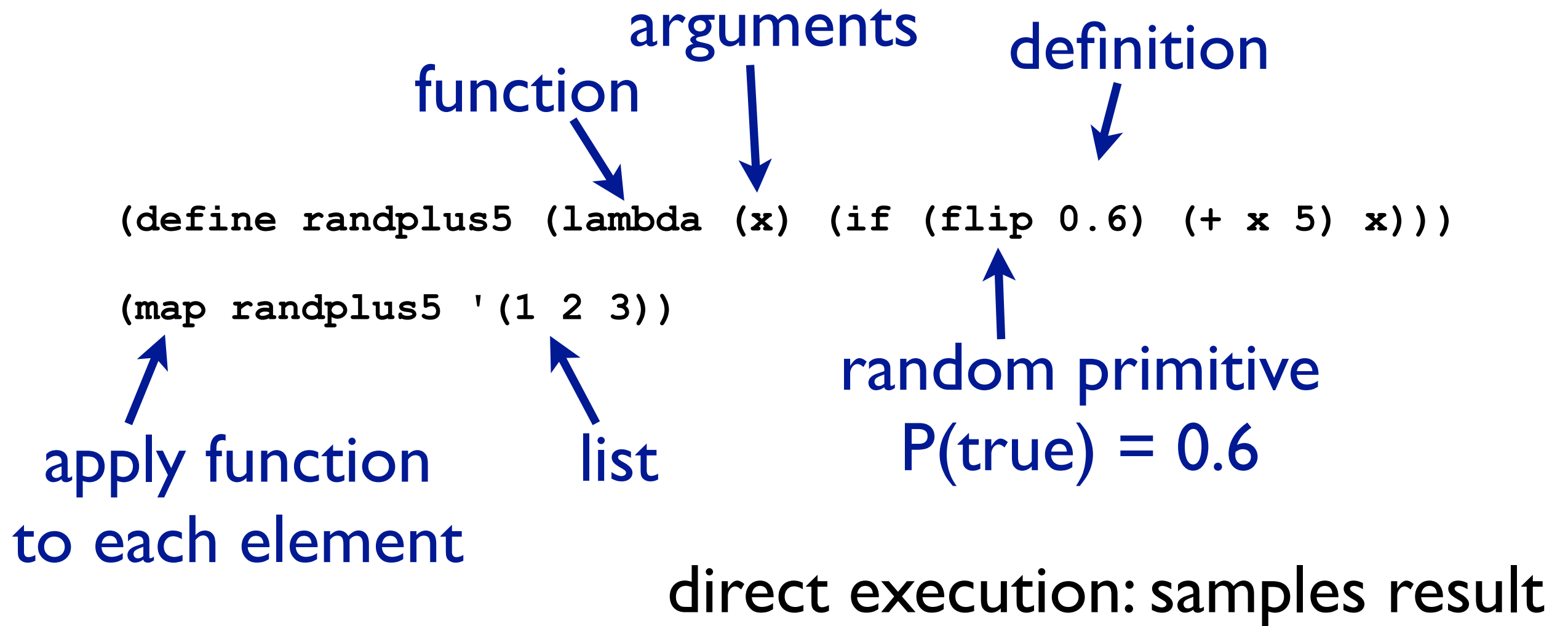
Church Example



Church Example



Church Example



sampling also supports continuous RVs, e.g.,
`(* (gaussian 0 1) (gaussian 0 1))`

Computing probability distribution

```
(enumeration-query  
  (define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))  
  (map randplus5 '(1 2))  
  true  
)
```

enumerates all executions &
sums probabilities per result

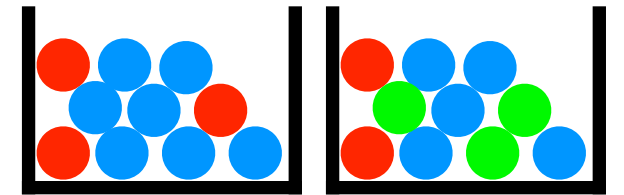
query

evidence

```
(( (1 2) (1 7) (6 2) (6 7)) (0.16 0.24 0.24 0.36))
```

Church by example:

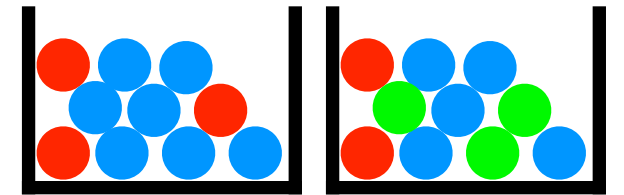
A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

Church by example:

A bit of gambling

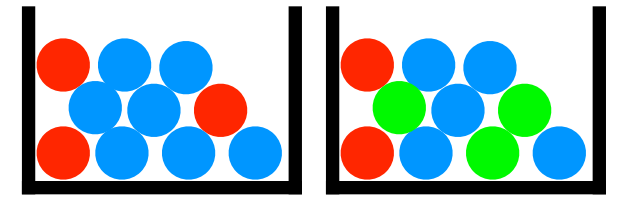


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))
```

Church by example:

A bit of gambling



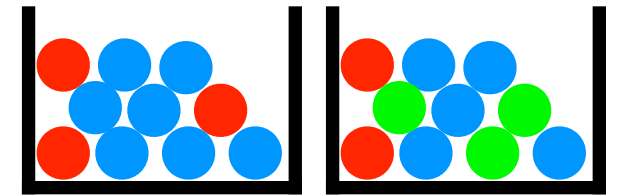
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))
```

```
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))
```


Church by example:

A bit of gambling

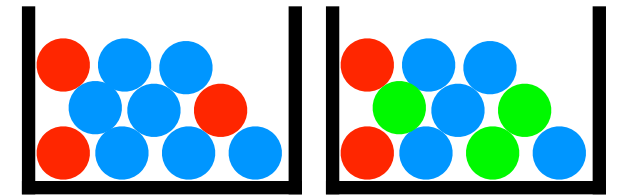


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5))))))
```

Church by example:

A bit of gambling

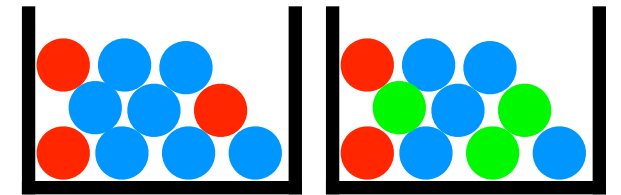


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5))))))  
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))
```

Church by example:

A bit of gambling

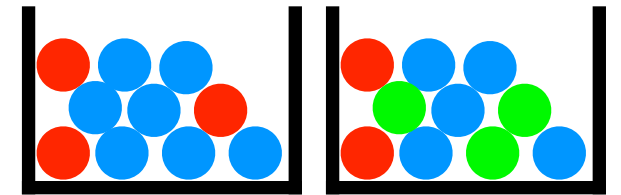


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5))))))  
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))  
(define win1 (and (heads) redball))
```

Church by example:

A bit of gambling

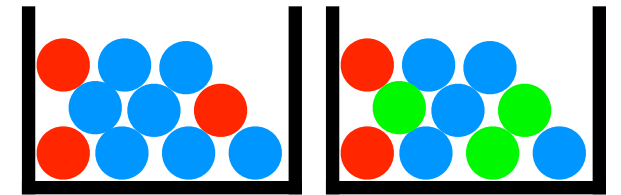


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))  
  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5)))))  
  
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))  
  
(define win1 (and (heads) redball))  
  
(define win2 (equal? (color1) (color2)))
```

Church by example:

A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))  
  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5)))))  
  
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))  
  
(define win1 (and (heads) redball))  
  
(define win2 (equal? (color1) (color2)))  
  
(define win (or win1 win2))
```

Probabilistic Programming Summary

- Church: functional programming + random primitives
- probabilistic generative model
- stochastic memoization
- sampling
- increasing number of probabilistic programming languages using various underlying paradigms

Roadmap

- Modeling (ProbLog and Church, another representative of PP)
- Inference
- Learning
- Dynamics and Decisions

... with some detours on the way

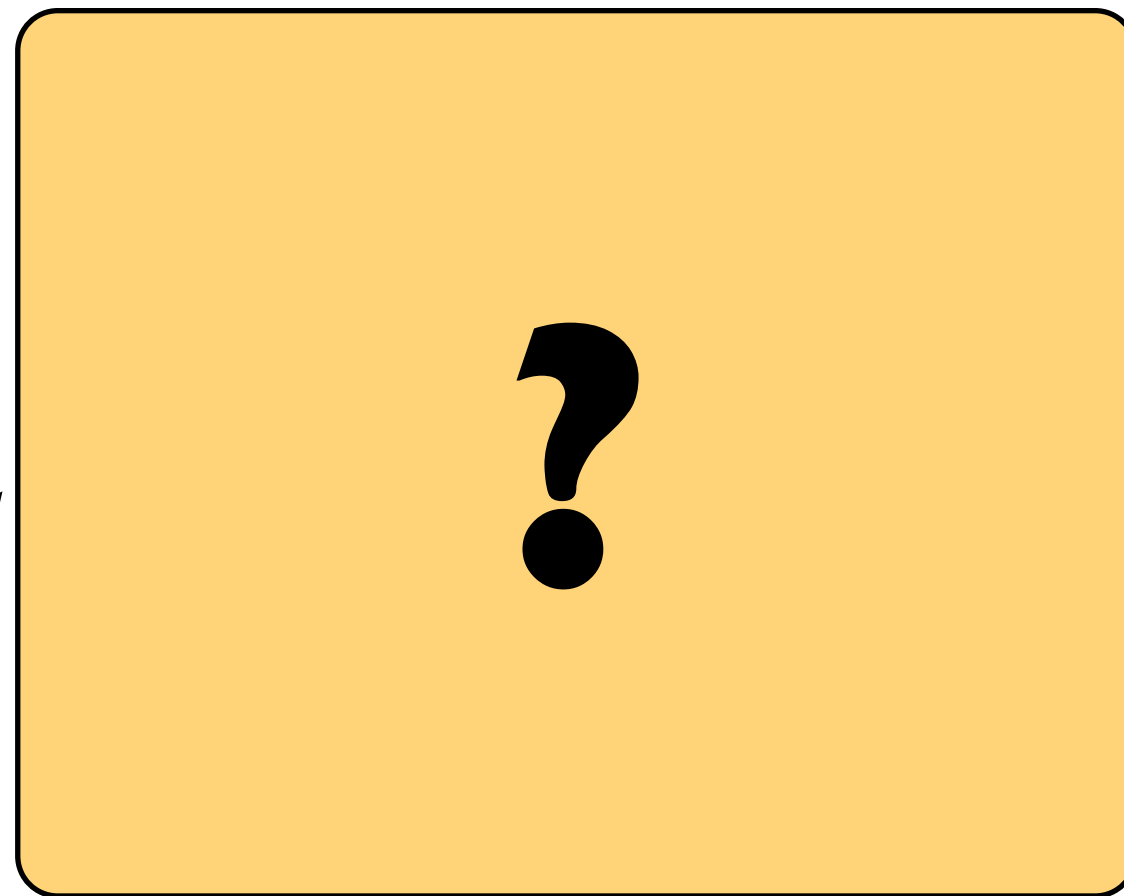
Answering Questions

Given:

program

queries

evidence



Find:

marginal
probabilities

conditional
probabilities

MPE state

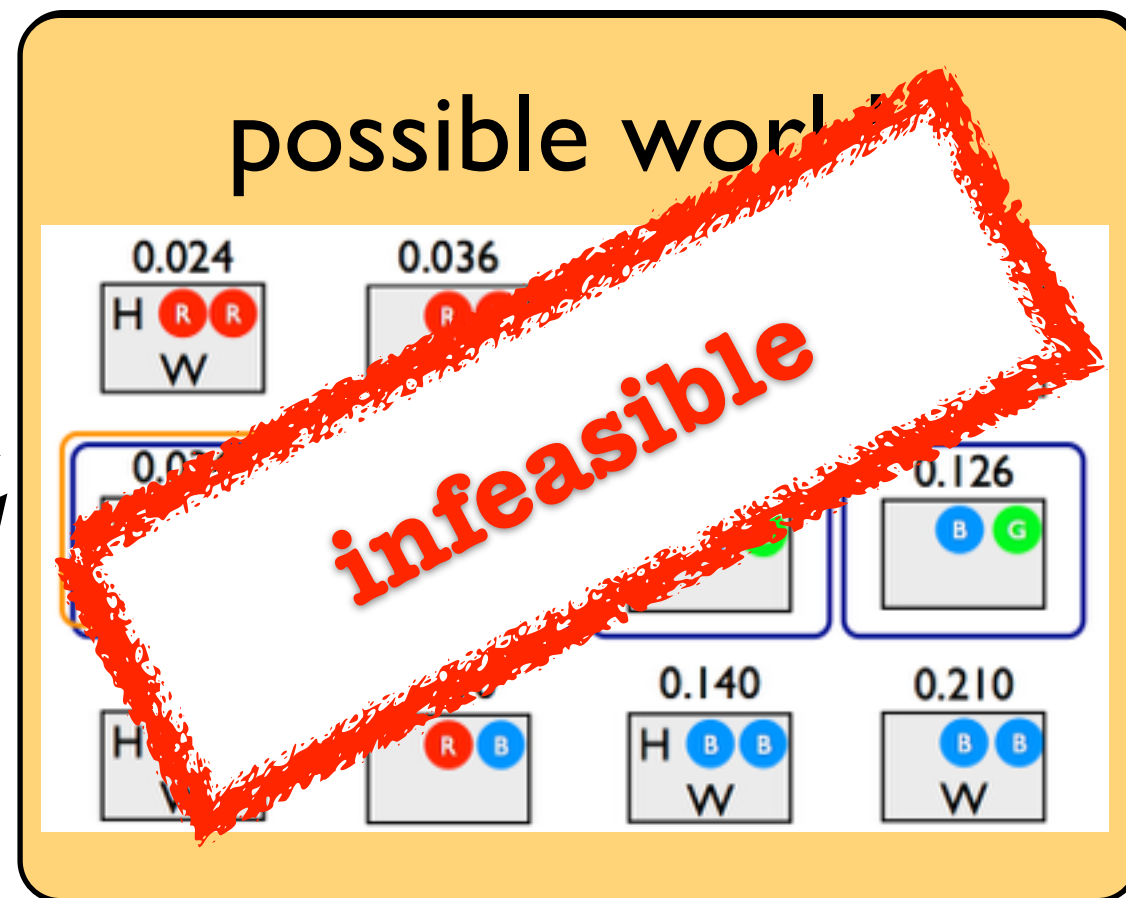
Answering Questions

Given:

program

queries

evidence



Find:

marginal probabilities

conditional probabilities

MPE state

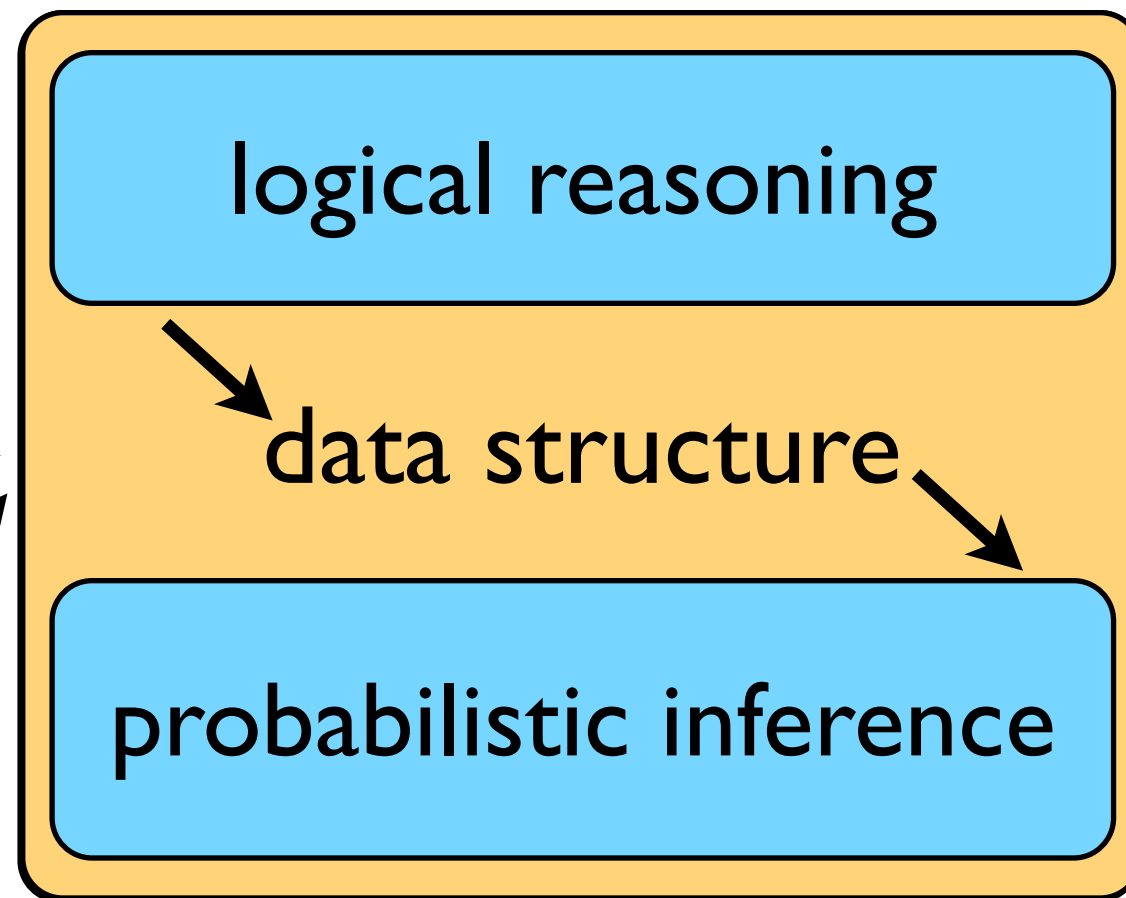
Answering Questions

Given:

program

queries

evidence



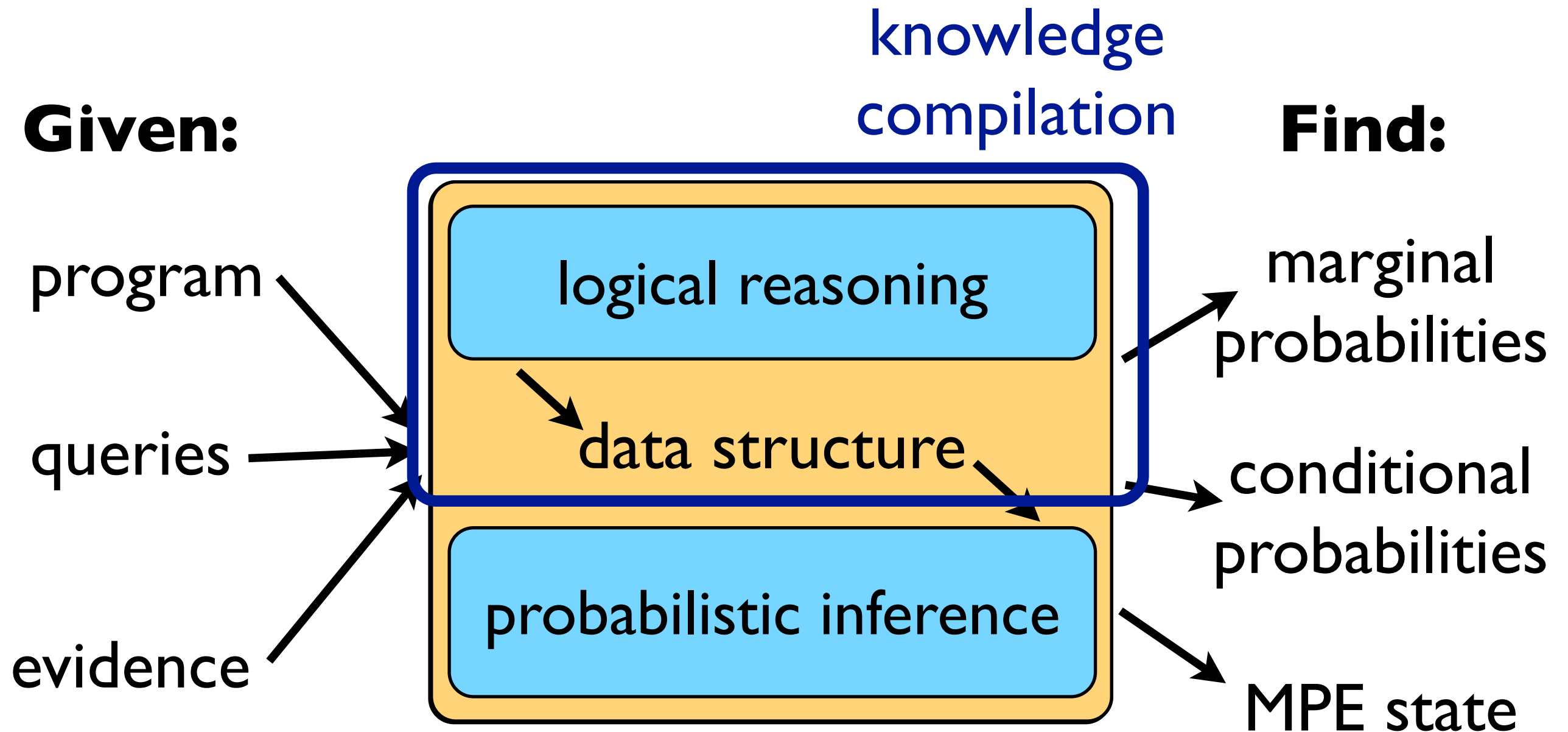
Find:

marginal
probabilities

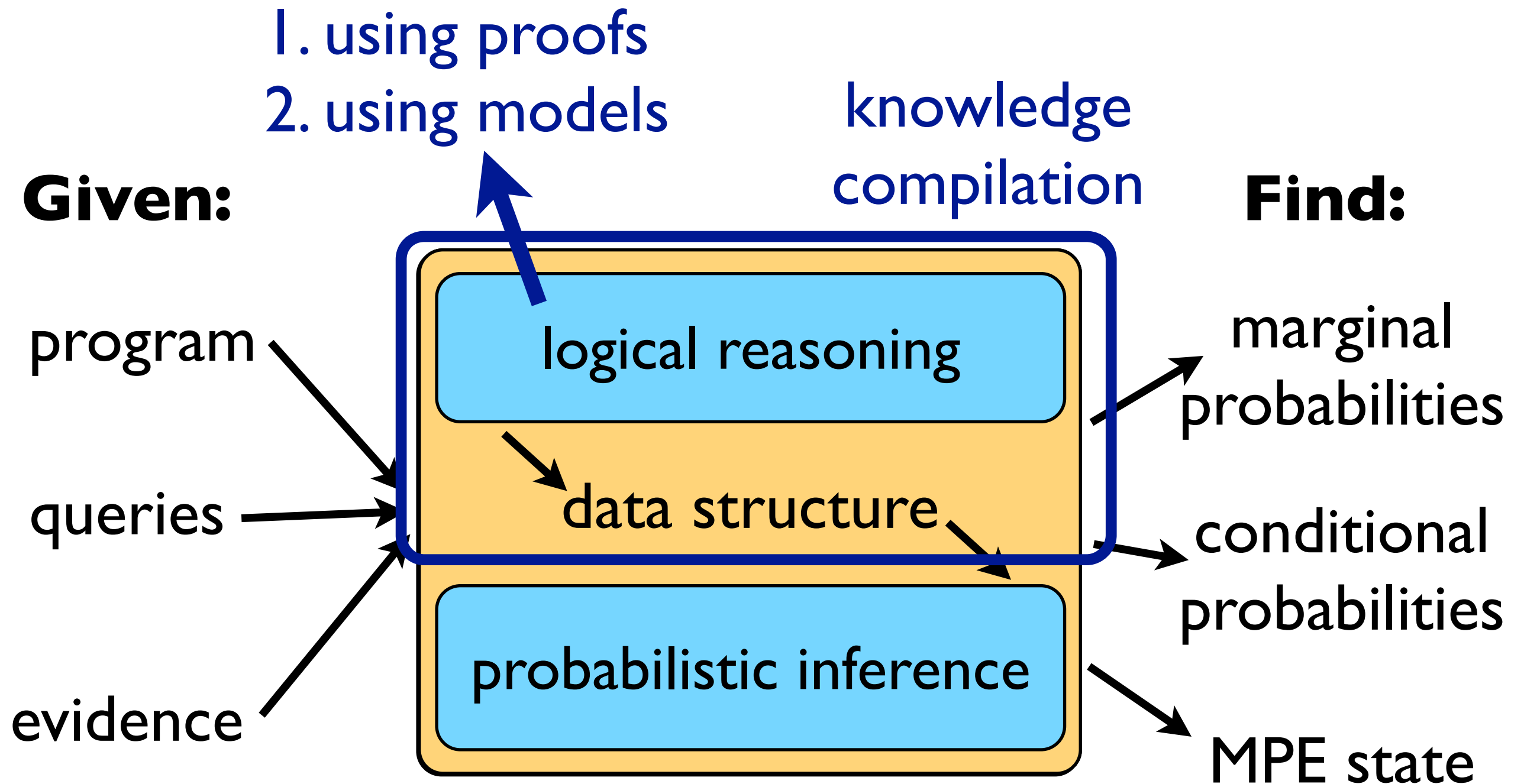
conditional
probabilities

MPE state

Answering Questions



Answering Questions



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .  
  
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

Logical Reasoning: Proofs in Prolog

```
?- smokes(carl) .
```

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .  
  
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

Logical Reasoning: Proofs in Prolog

`?- smokes(carl) .`

`?- stress(carl) .`



```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .  
  
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

```
      ?- smokes(carl) .  
     /  \  
    /      \  
?- stress(carl) .    ?- influences(Y,carl) , smokes(Y) .
```

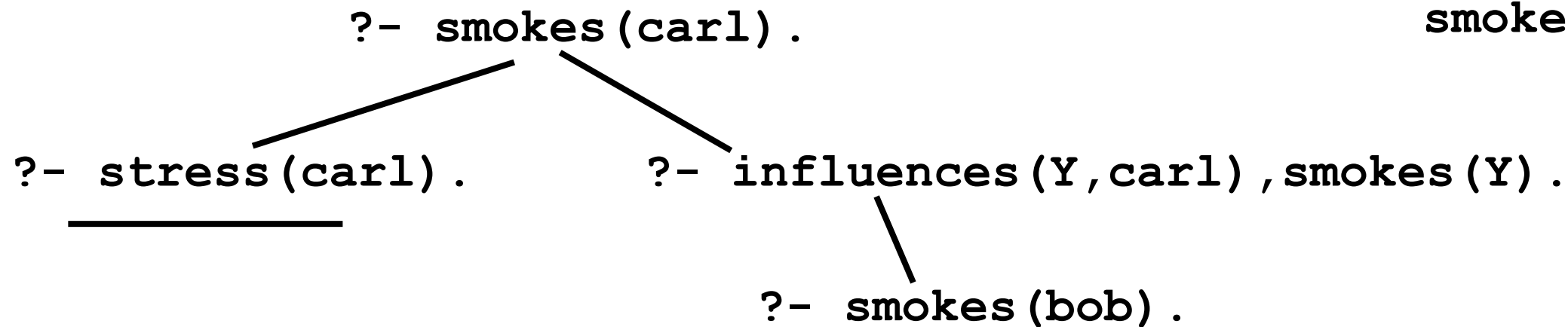

Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .  
  
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

```
      ?- smokes(carl) .  
     /  \  
    /      \  
?- stress(carl) .    ?- influences(Y,carl) , smokes(Y) .  
  _____
```

Logical Reasoning: Proofs in Prolog

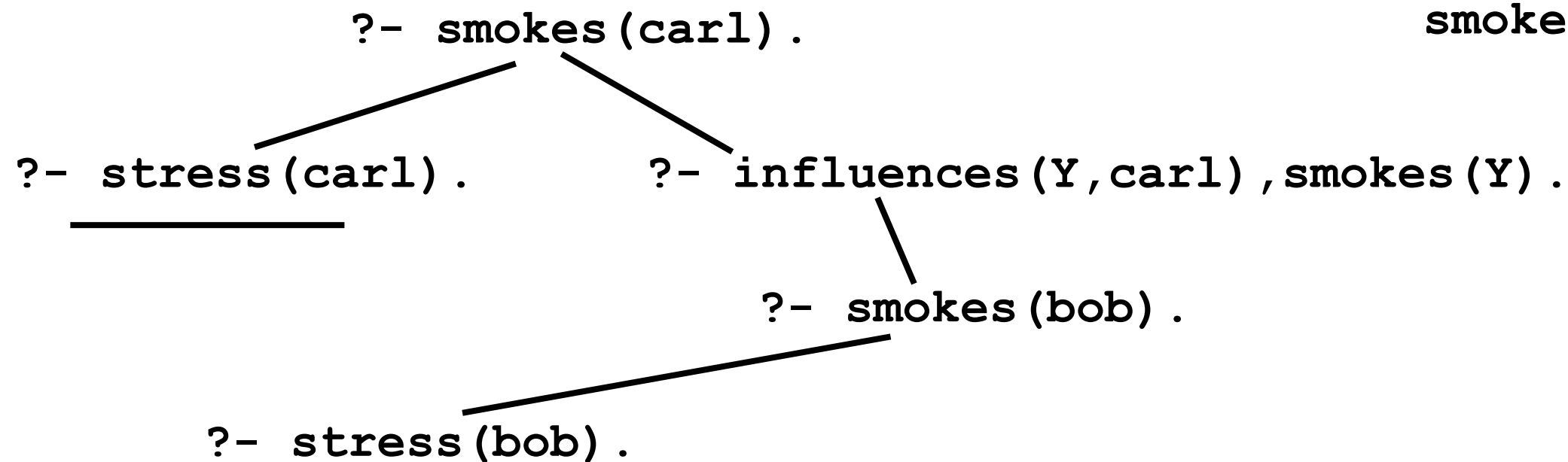
```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .  
  
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

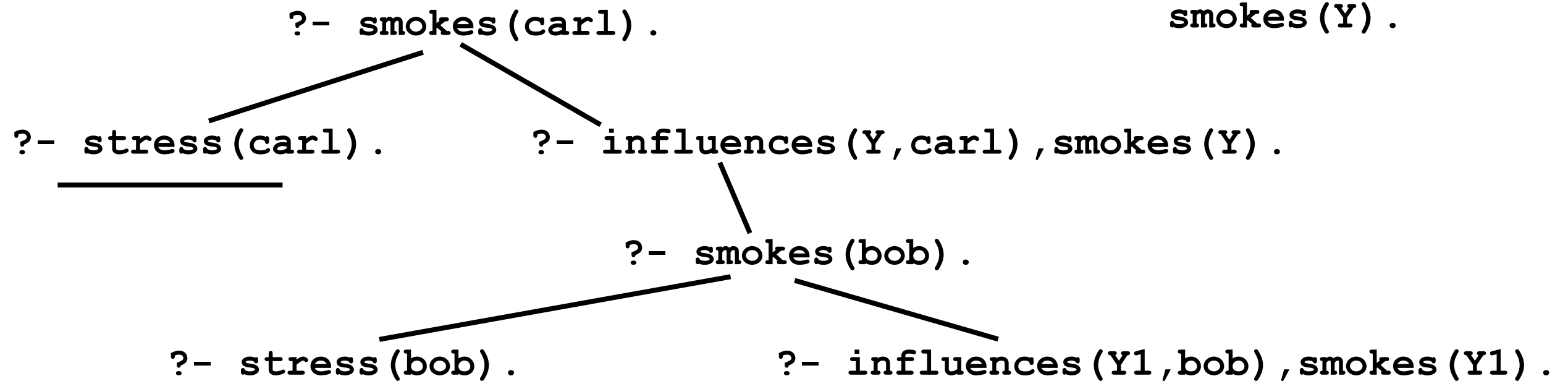
```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



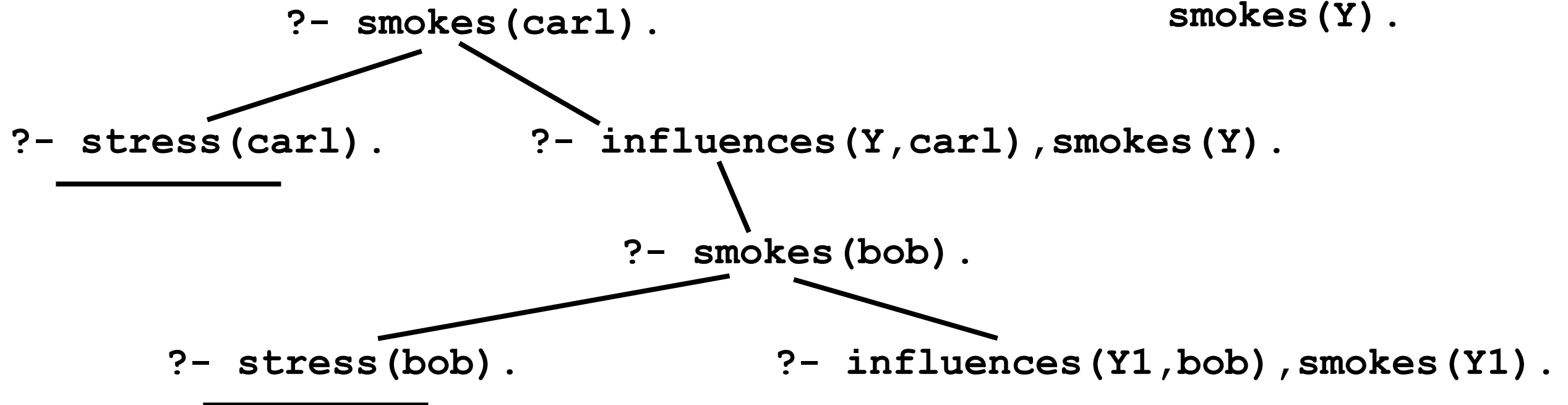
Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .  
  
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

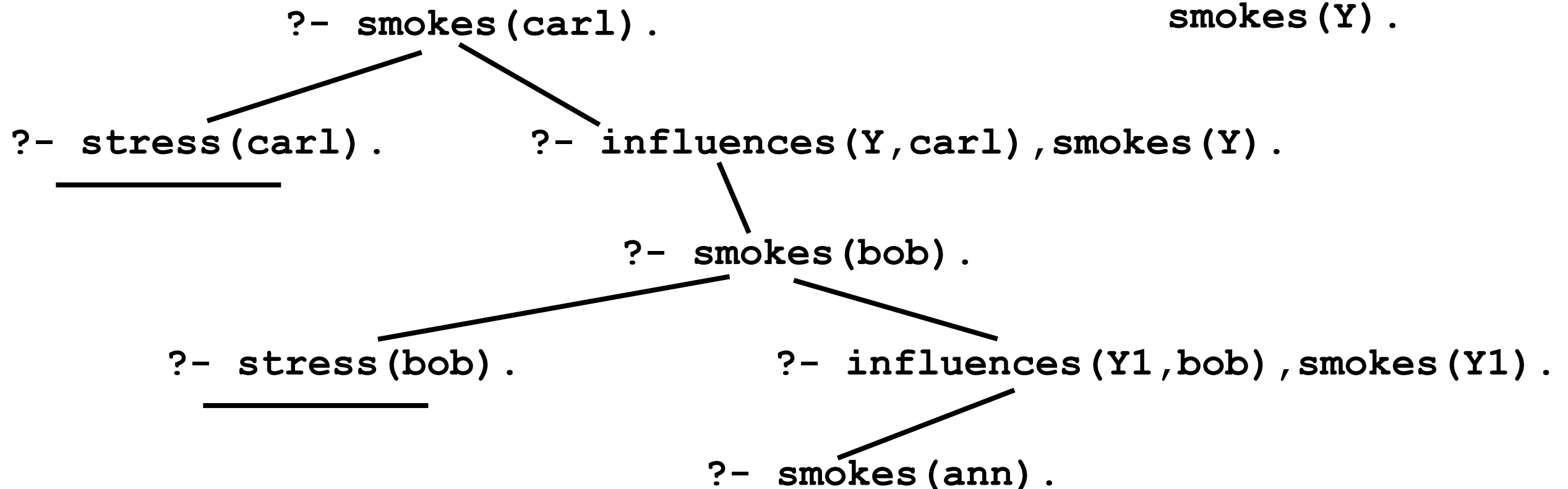
```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .  
  
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

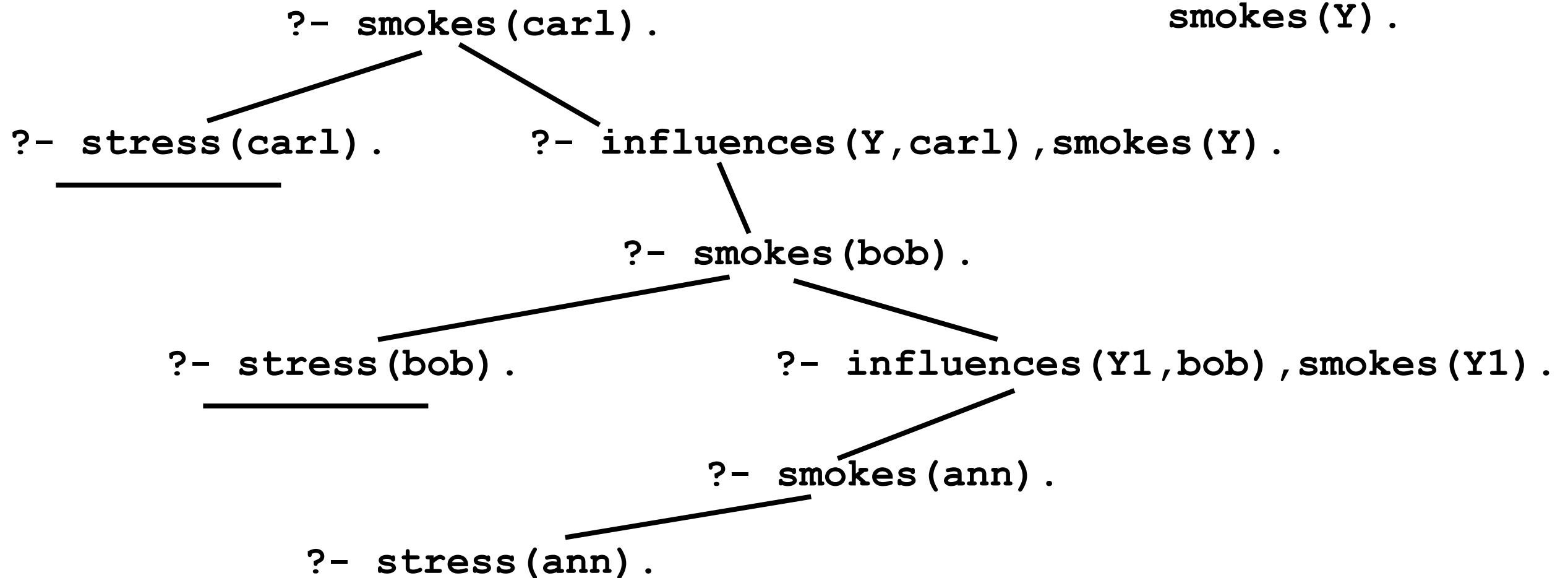
```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

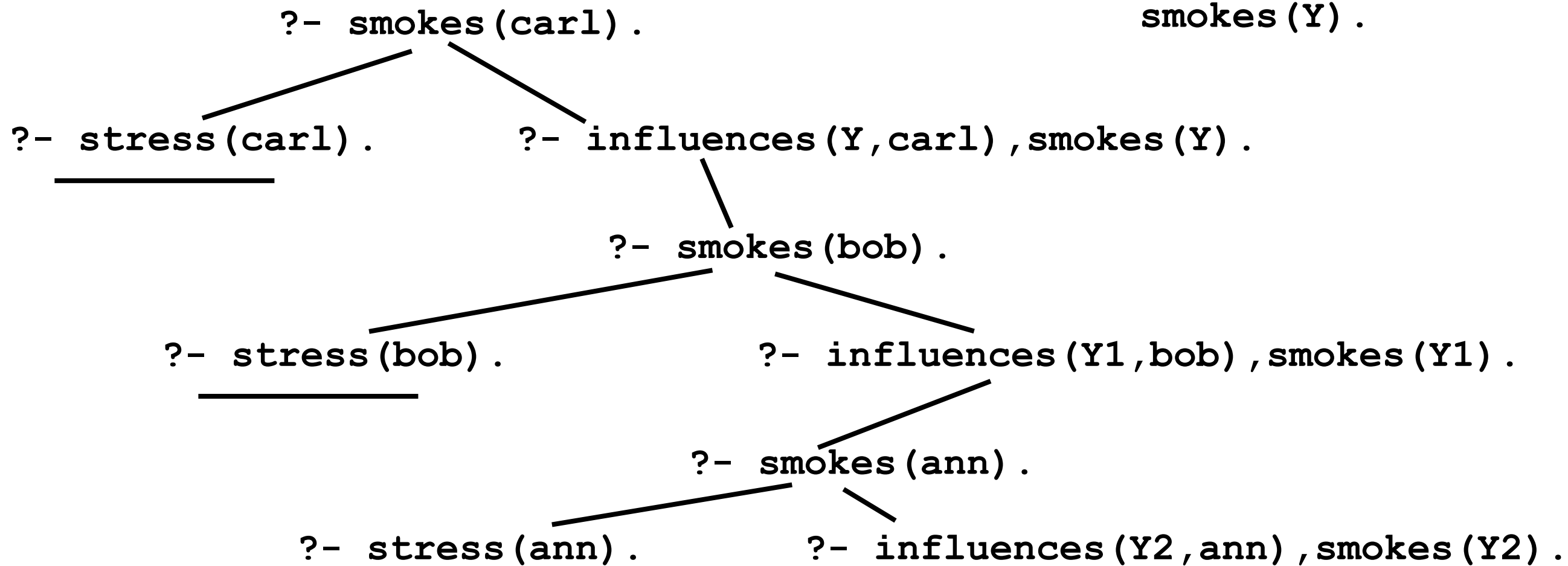
```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

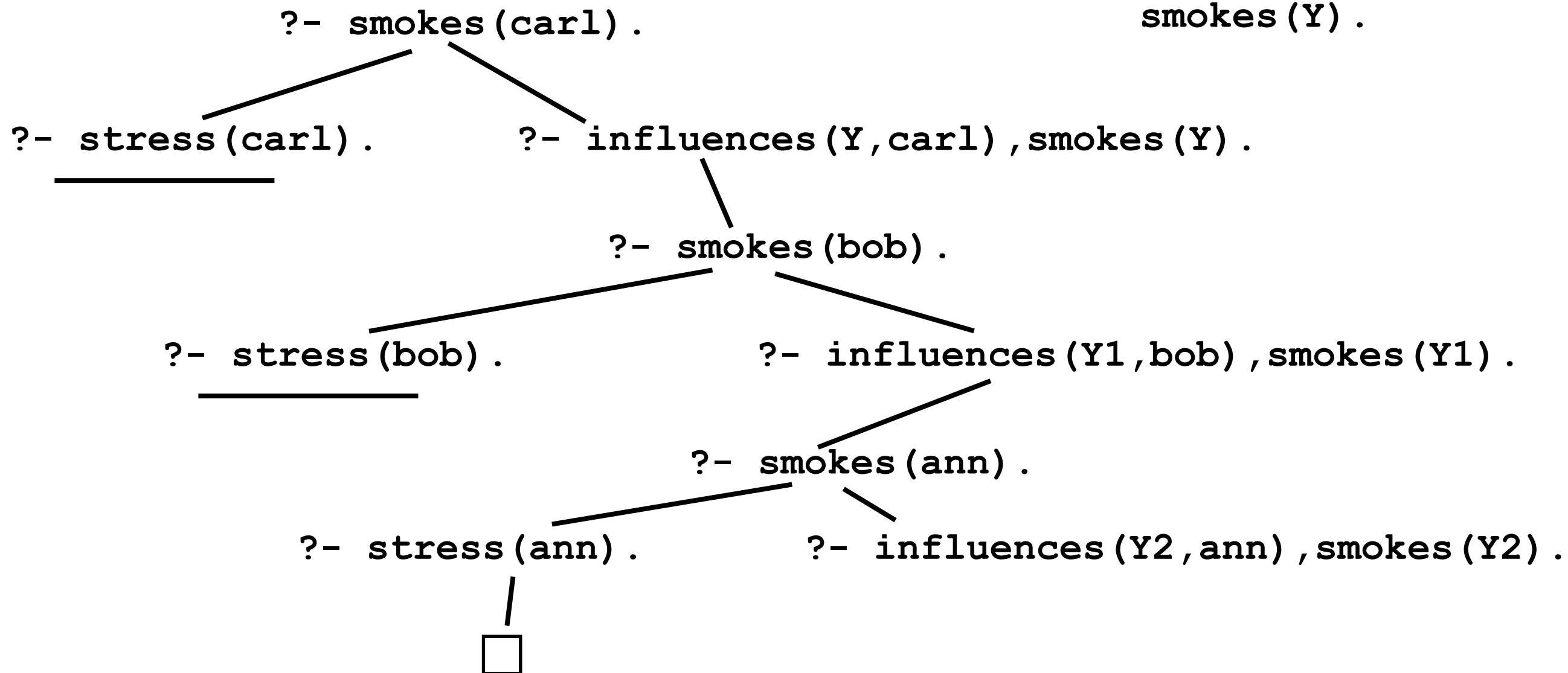
```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

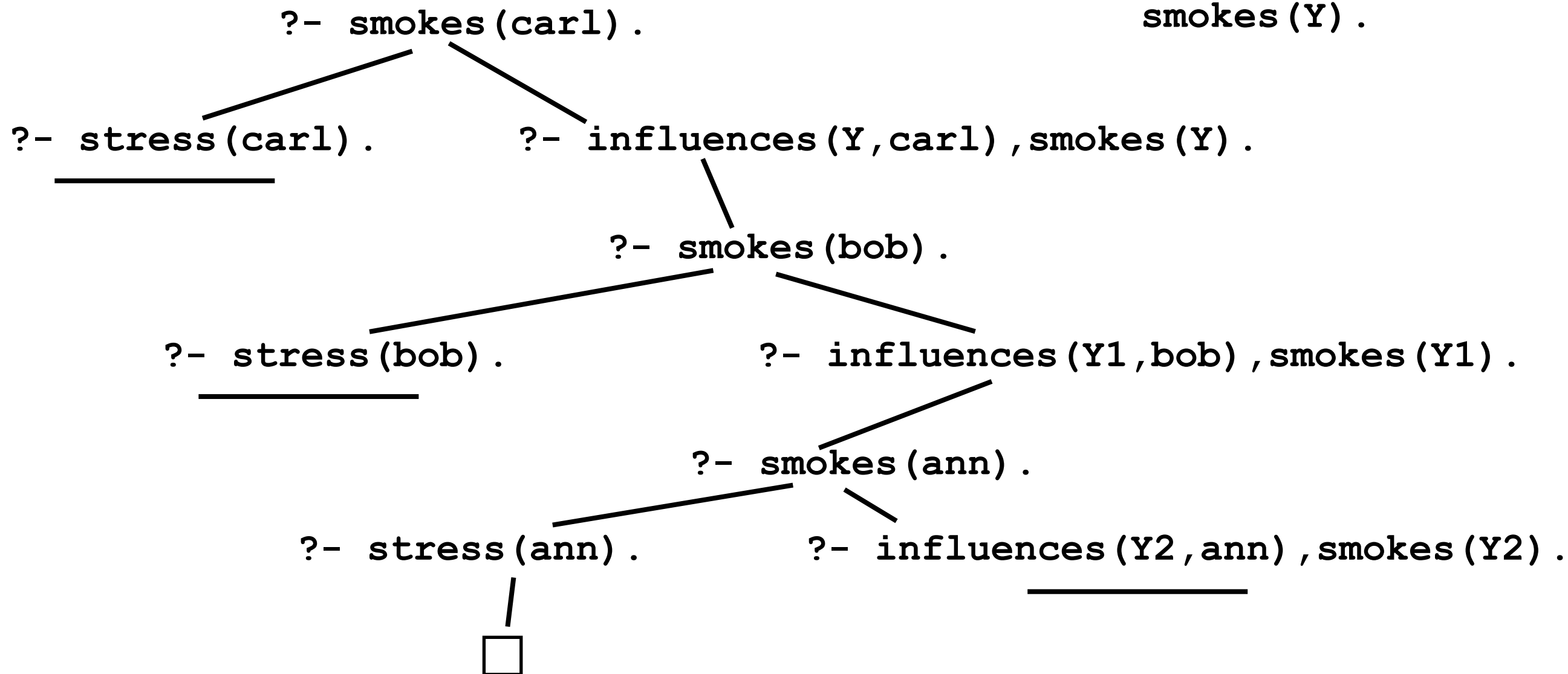
```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

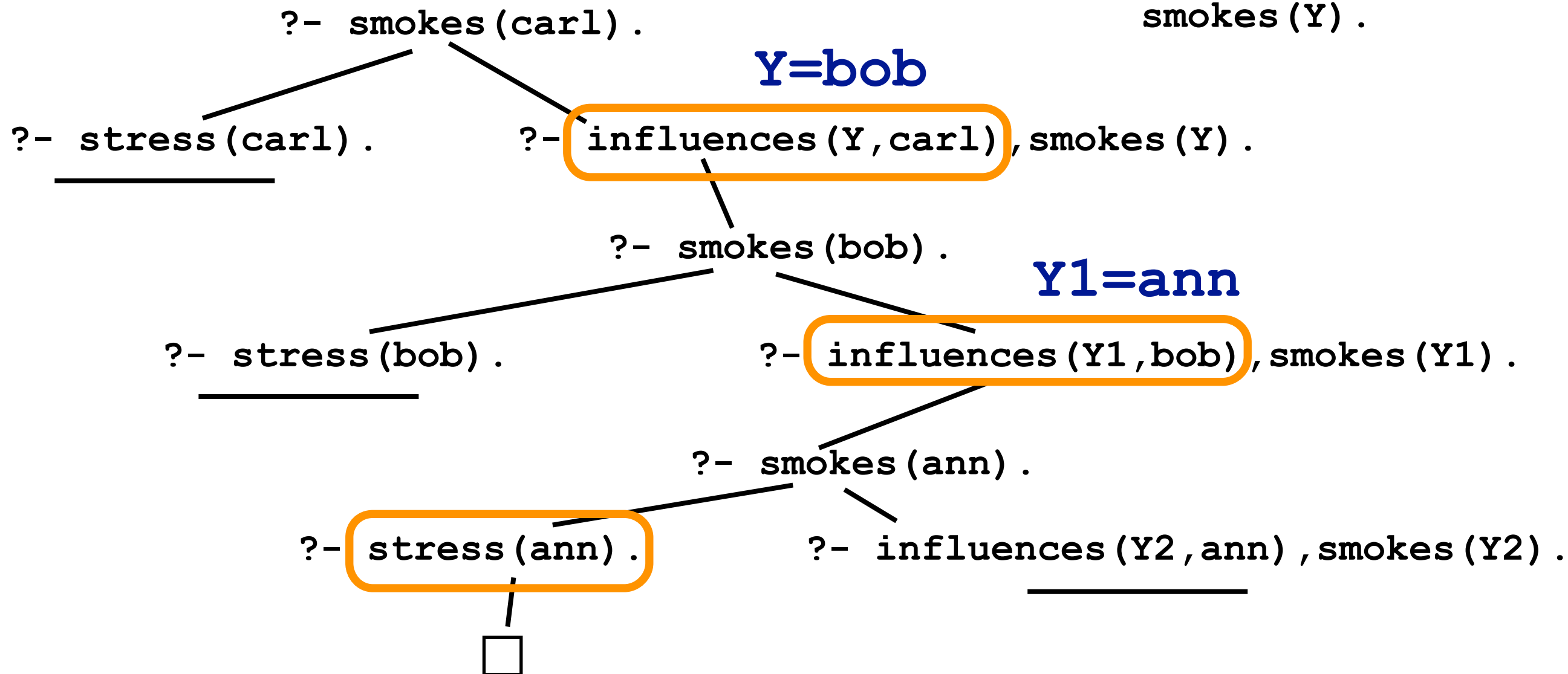
```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

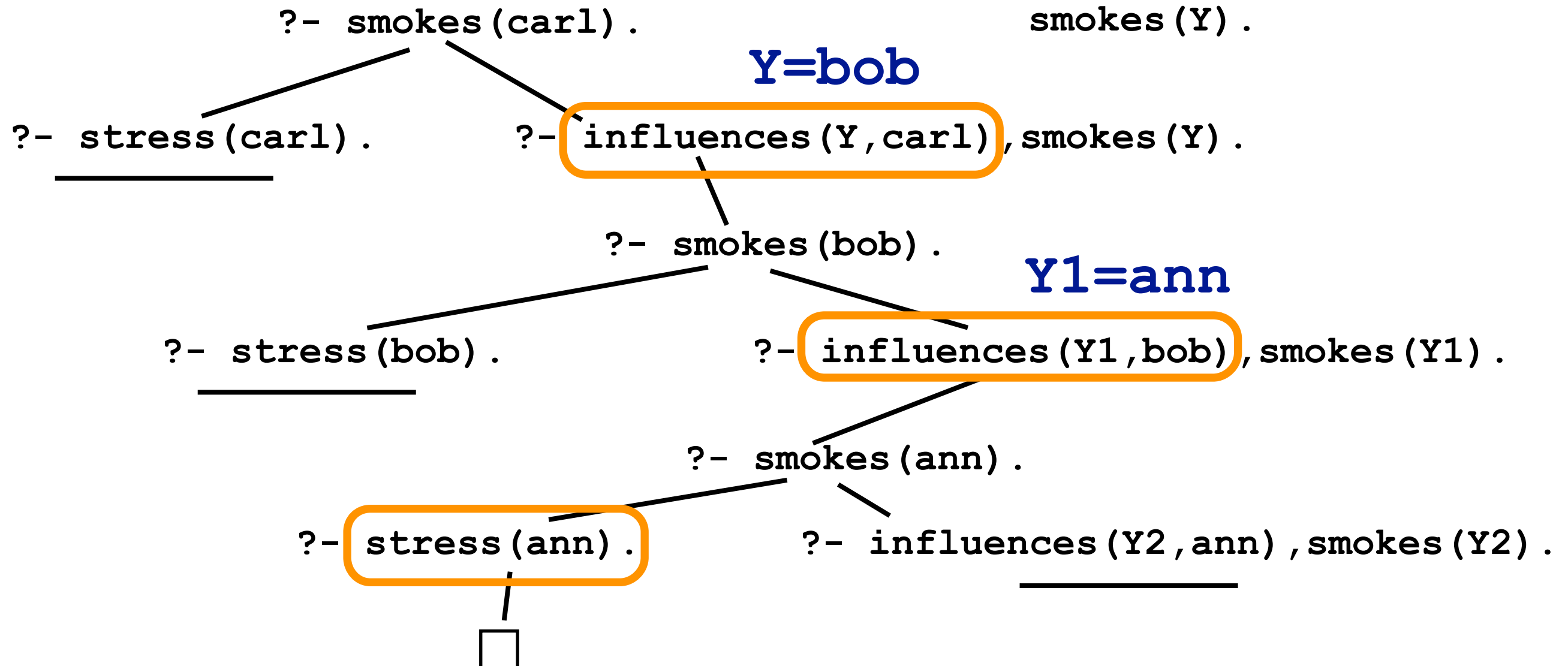


proof = facts used in successful derivation:
`influences(bob,carl) & influences(ann,bob) & stress(ann)`

Proofs in ProbLog

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

```
smokes(X) :- stress(X).
smokes(X) :-
    influences(Y,X),
    smokes(Y).
```



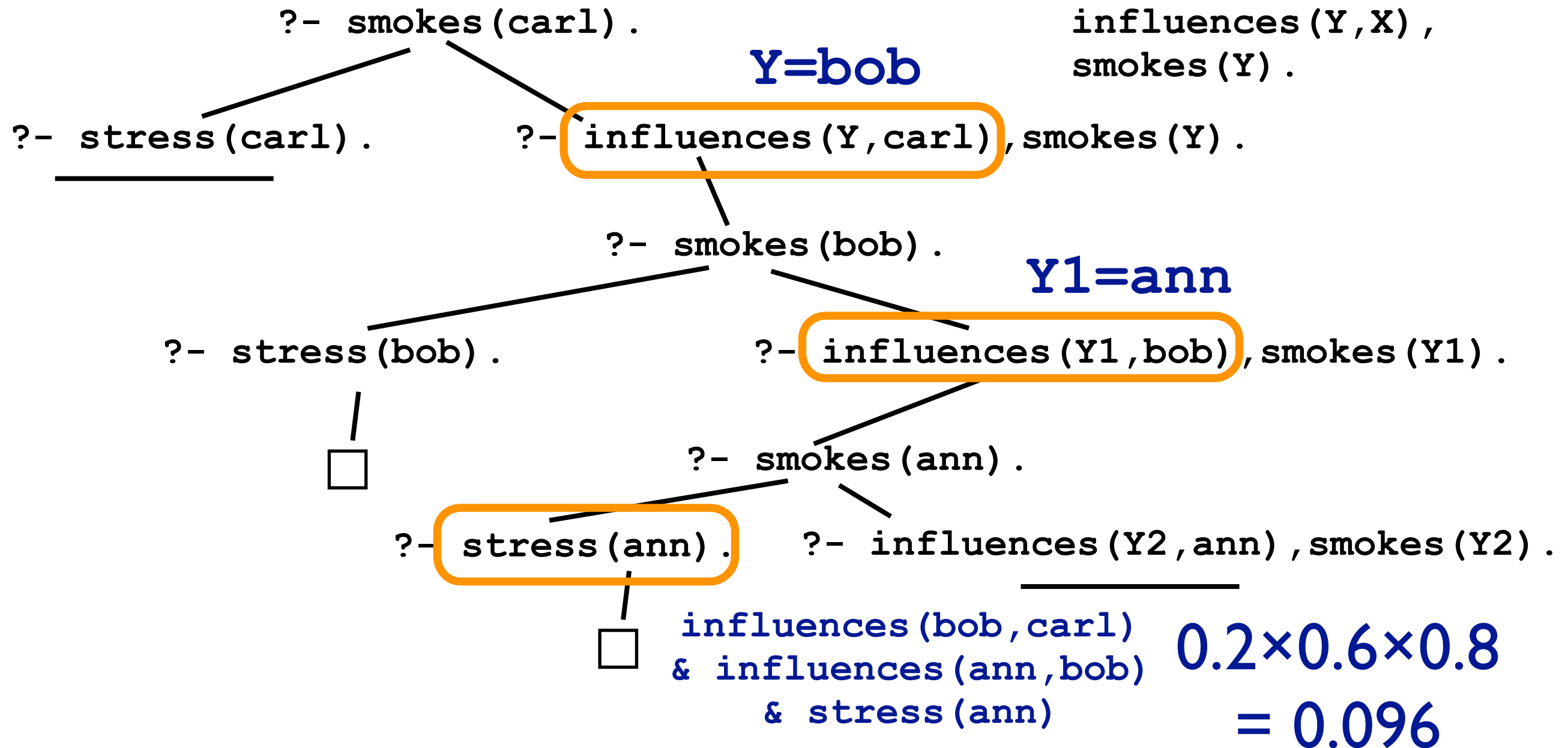
$\text{influences}(\text{bob}, \text{carl}) \ \& \ \text{influences}(\text{ann}, \text{bob}) \ \& \ \text{stress}(\text{ann})$

probability of proof = $0.2 \times 0.6 \times 0.8 = 0.096$

Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

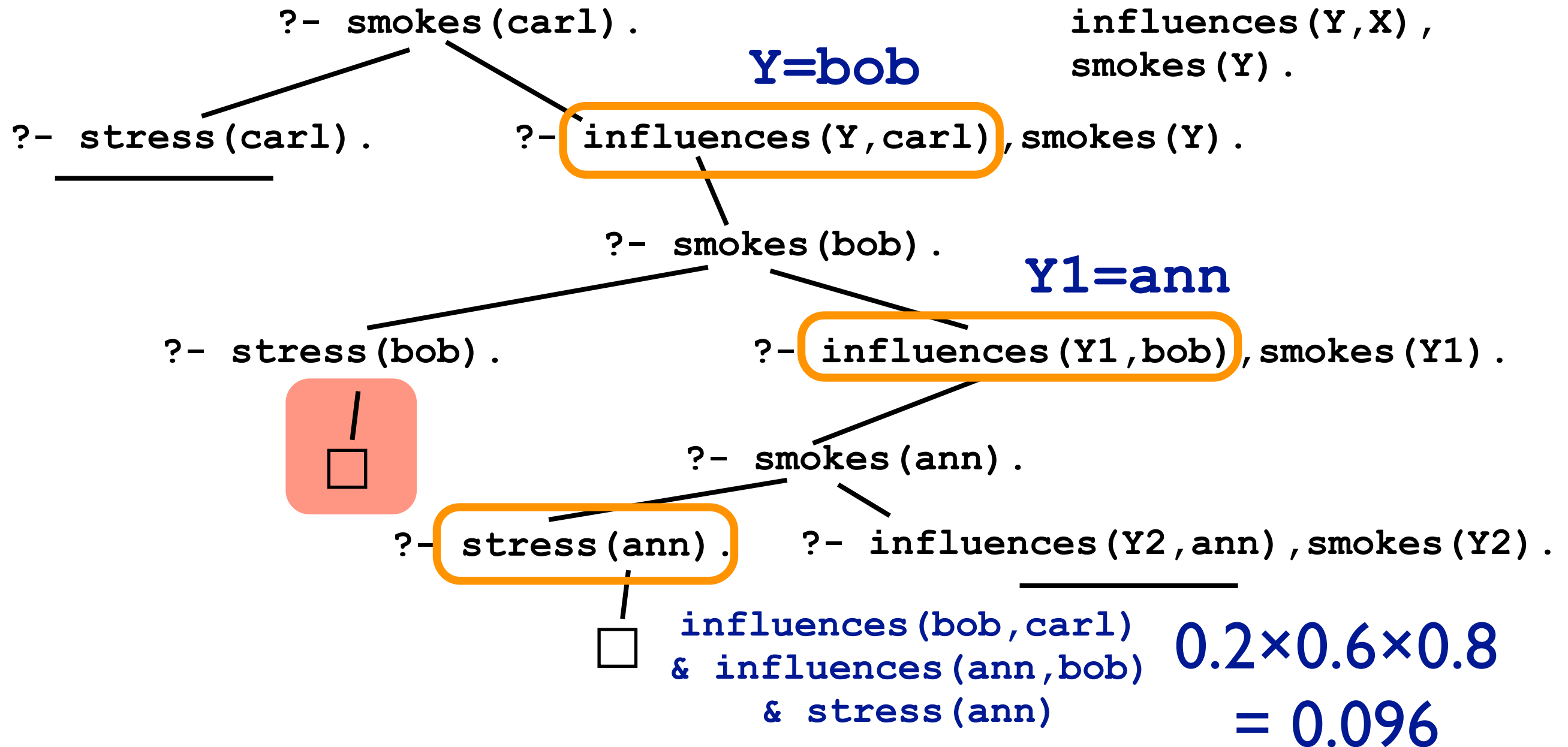
```
smokes(X) :- stress(X).
smokes(X) :-
    influences(Y,X),
    smokes(Y).
```



Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

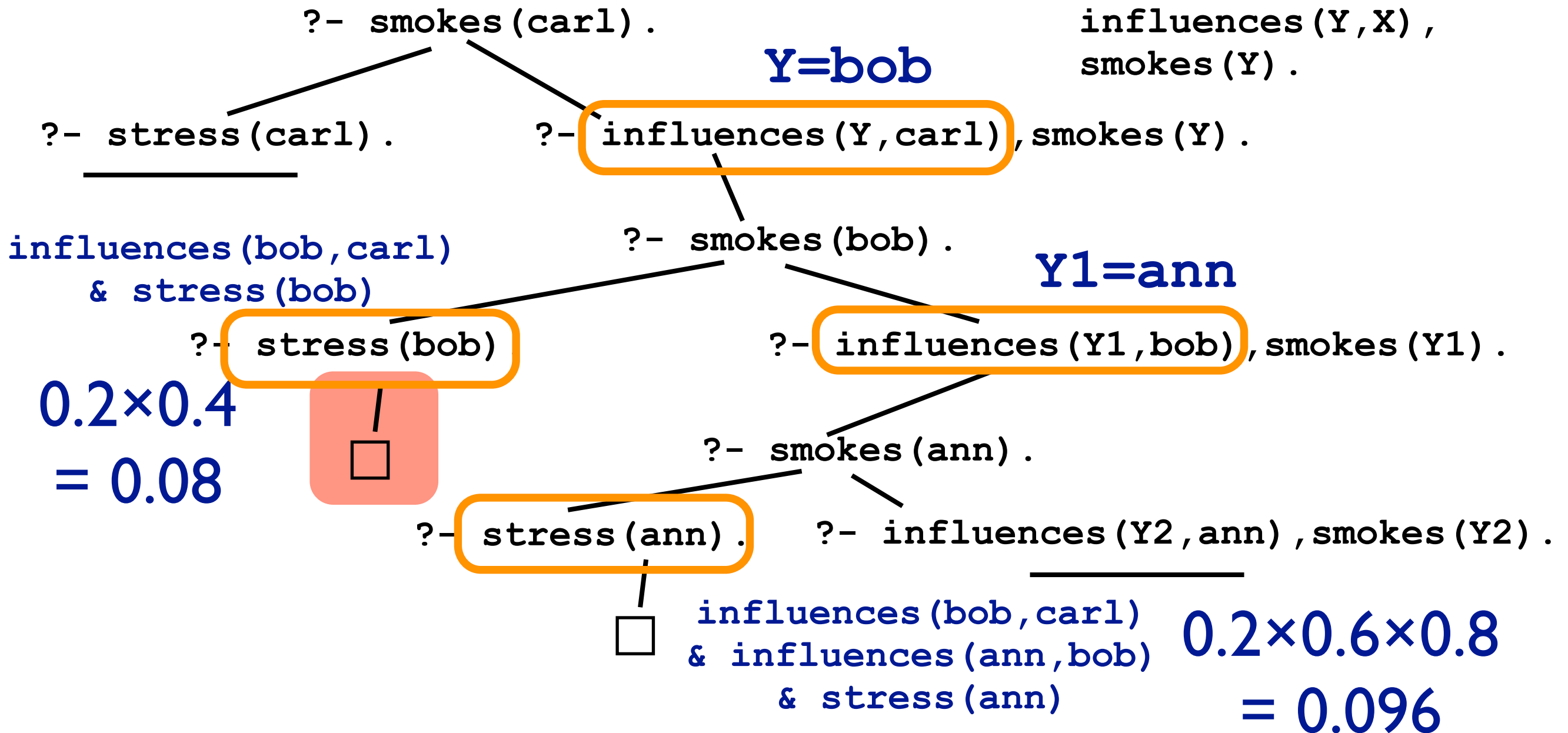
```
smokes(X) :- stress(X).
smokes(X) :-
    influences(Y,X),
    smokes(Y).
```



Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

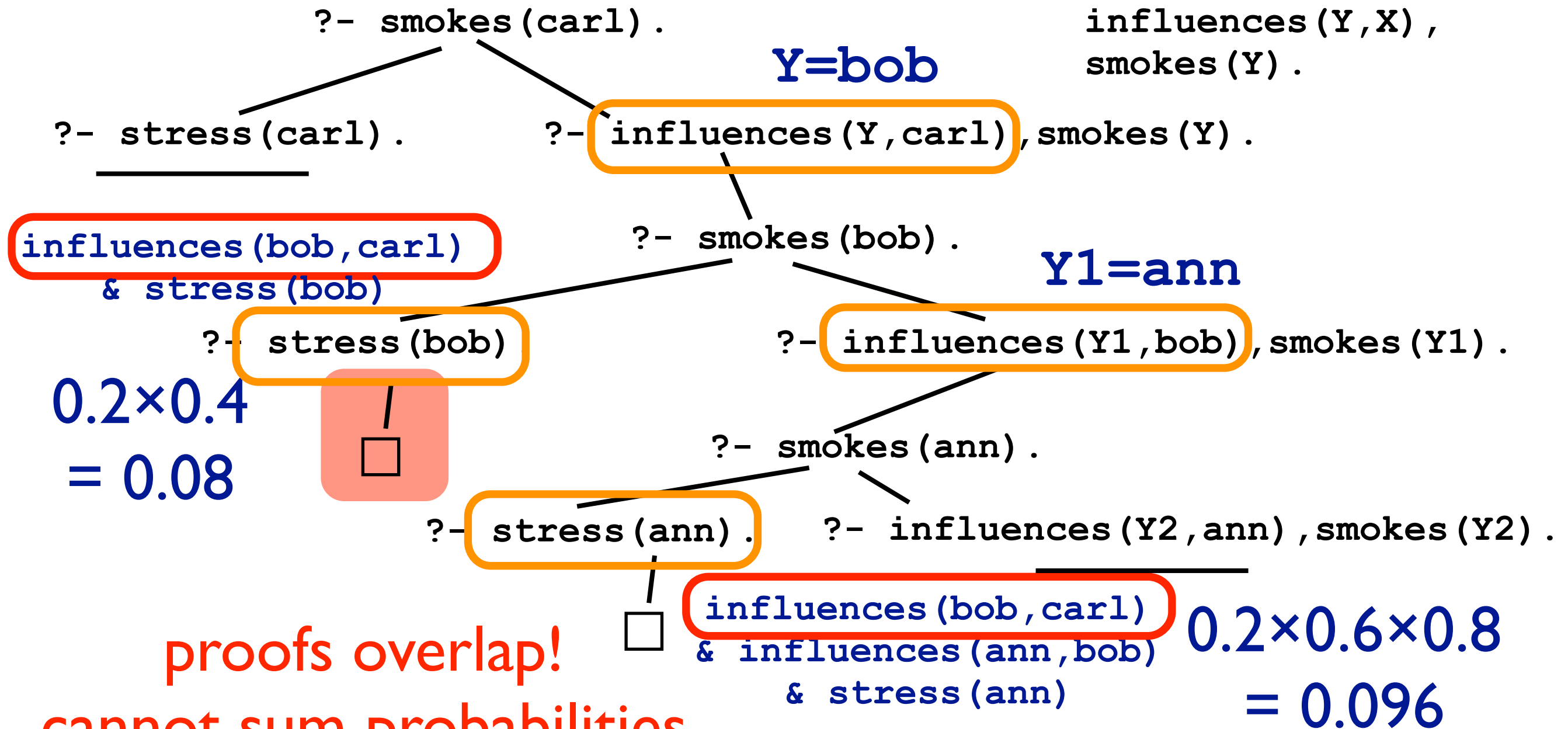
```
smokes(X) :- stress(X).
smokes(X) :-
    influences(Y,X),
    smokes(Y).
```



Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

```
smokes(X) :- stress(X).
smokes(X) :-
    influences(Y,X),
    smokes(Y).
```



proofs overlap!
cannot sum probabilities
(disjoint-sum-problem)

Disjoint-Sum-Problem

possible worlds

`infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)`

`...`

Disjoint-Sum-Problem

possible worlds

`influences(bob,carl) &
influences(ann,bob) & stress(ann)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)`

...

Disjoint-Sum-Problem

possible worlds

`influences(bob,carl) &
influences(ann,bob) & stress(ann)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)`

`... influences(bob,carl) & stress(bob)`

Disjoint-Sum-Problem

possible worlds

`influences(bob,carl) &
influences(ann,bob) & stress(ann)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)`

`... influences(bob,carl) & stress(bob)`

sum of proof probabilities: $0.096 + 0.08 = 0.1760$

Disjoint-Sum-Problem

possible worlds

`influences(bob,carl) &
influences(ann,bob) & stress(ann)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)`

0.0576

`infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)`

0.0384

`infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)`

0.0256

`infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)`

0.0096

`infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)`

0.0064

...

`influences(bob,carl) & stress(bob)`

$\Sigma = 0.1376$

sum of proof probabilities: $0.096 + 0.08 = 0.1760$

Disjoint-Sum-Problem

possible worlds

solution: knowledge compilation

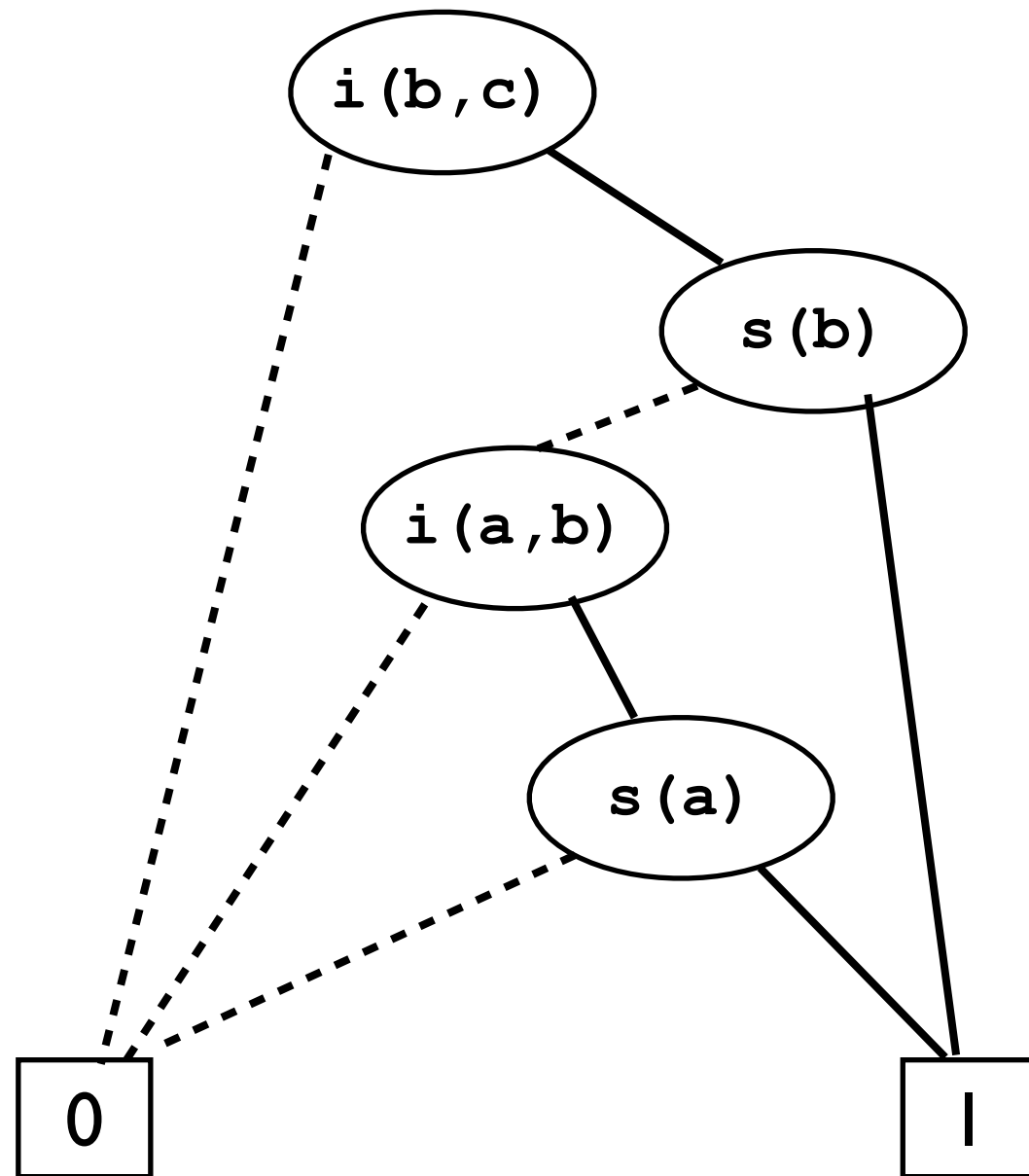
<code>infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)</code>	0.0576
<code>infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)</code>	0.0384
<code>infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)</code>	0.0256
<code>infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)</code>	0.0096
<code>infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)</code>	0.0064

... `influences(bob,carl) & stress(bob)` $\Sigma = 0.1376$

sum of proof probabilities: $0.096 + 0.08 = 0.1760$

Binary Decision Diagrams [Bryant 86]

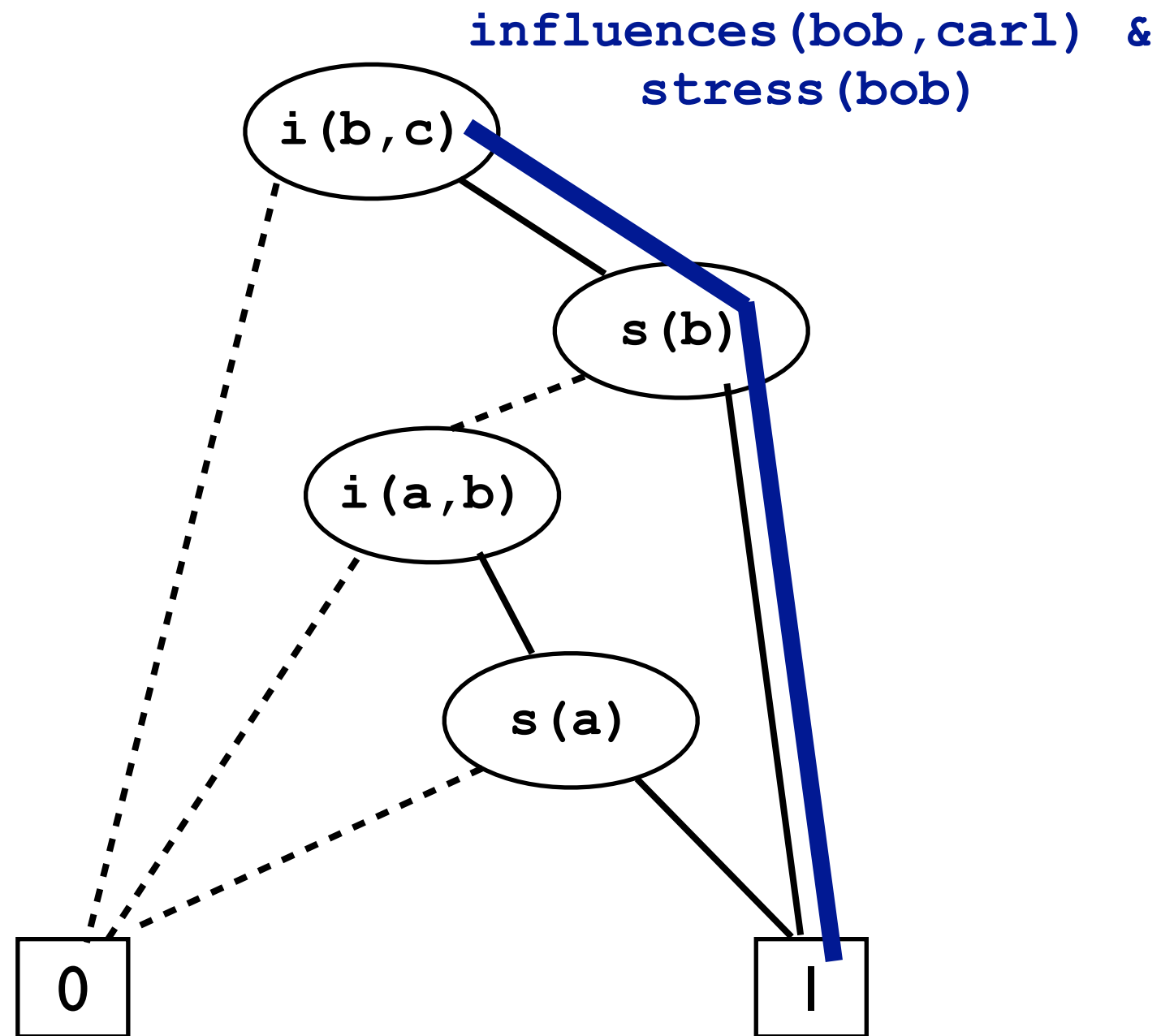
- compact graphical representation of Boolean formula
- automatically disjoins proofs
- popular in many branches of CS



Binary Decision Diagrams

[Bryant 86]

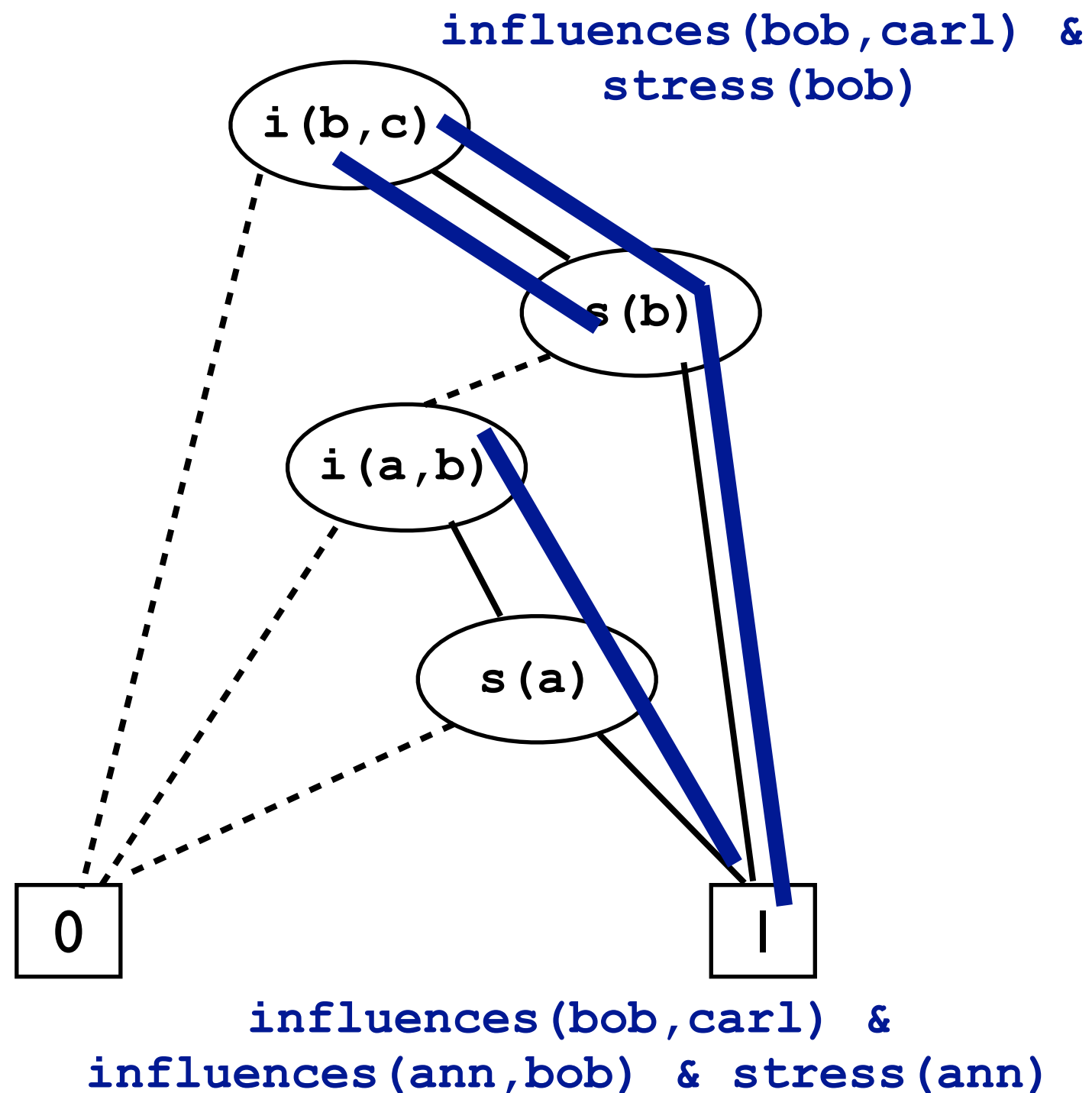
- compact graphical representation of Boolean formula
- automatically disjoins proofs
- popular in many branches of CS



Binary Decision Diagrams

[Bryant 86]

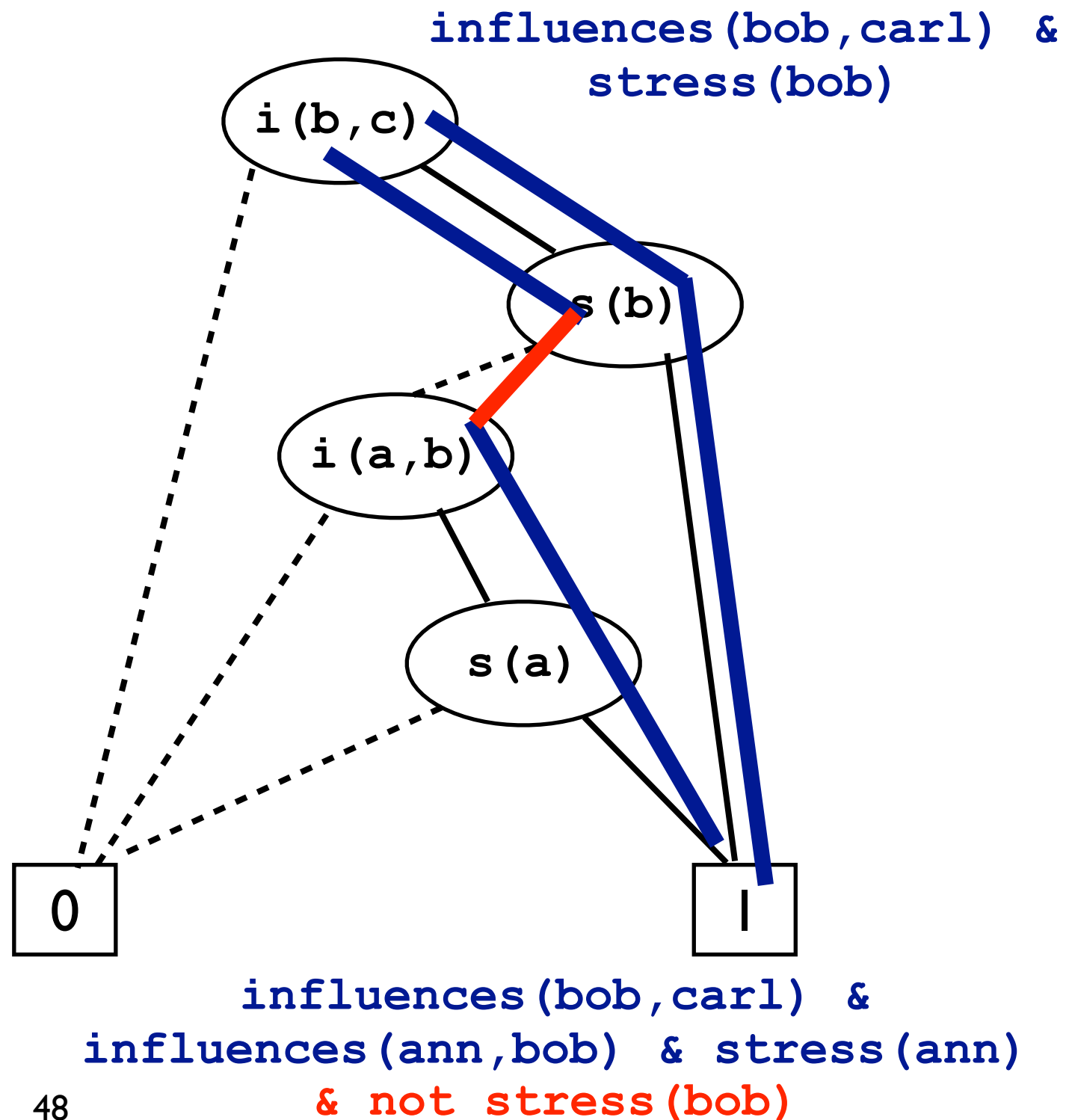
- compact graphical representation of Boolean formula
- automatically disjoins proofs
- popular in many branches of CS



Binary Decision Diagrams

[Bryant 86]

- compact graphical representation of Boolean formula
- automatically disjoins proofs
- popular in many branches of CS



Binary Decision Diagrams

[Bryant 86]

- compact graphical representation of Boolean formula
- popular in many branches of CS
- automatically disjoins proofs
→ efficient probability computation
- other representations exist (SDDs, d-DNNFs)
- knowledge compilation is state of the art for probabilistic inference (Darwiche et al.)

Initial Approach

(ProbLogI & others)

Find all proofs of query

Binary Decision
Diagram (BDD)

calculate marginal by
dynamic programming

Initial Approach

(ProbLogI & others)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),heads(3).
```

win

Find all proofs of query

Binary Decision
Diagram (BDD)

calculate marginal by
dynamic programming

Initial Approach

(ProbLogI & others)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),heads(3).
```

win

Find all proofs of query

Binary Decision
Diagram (BDD)

calculate marginal by
dynamic programming

```
heads(1)  
heads(2) & heads(3)
```

Initial Approach

(ProbLogI & others)

```
0.4::heads(1) .  
0.7::heads(2) .  
0.5::heads(3) .  
win :- heads(1) .  
win :- heads(2),heads(3) .
```

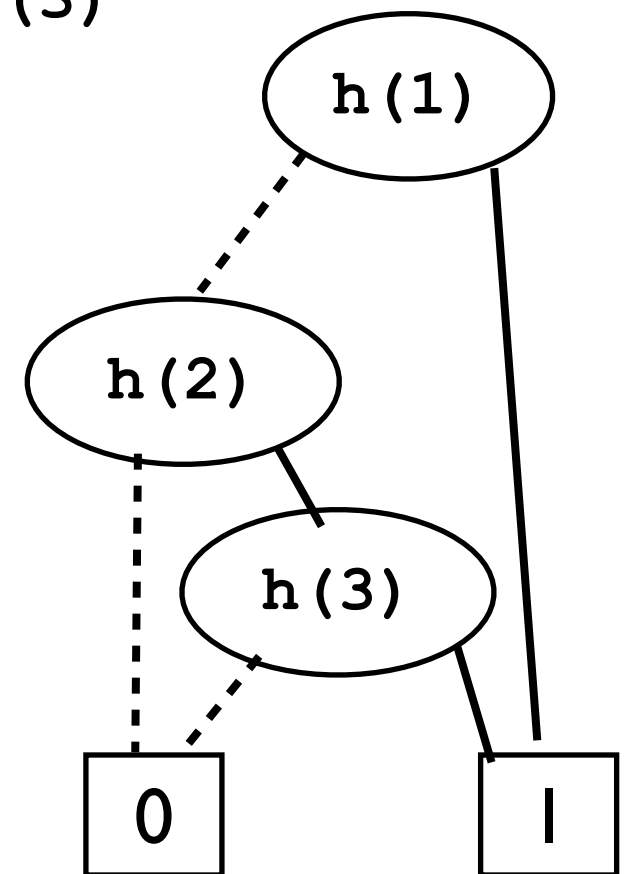
win

Find all proofs of query

Binary Decision
Diagram (BDD)

calculate marginal by
dynamic programming

```
heads(1)  
heads(2) & heads(3)
```



Initial Approach

(ProbLogI & others)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),heads(3).
```

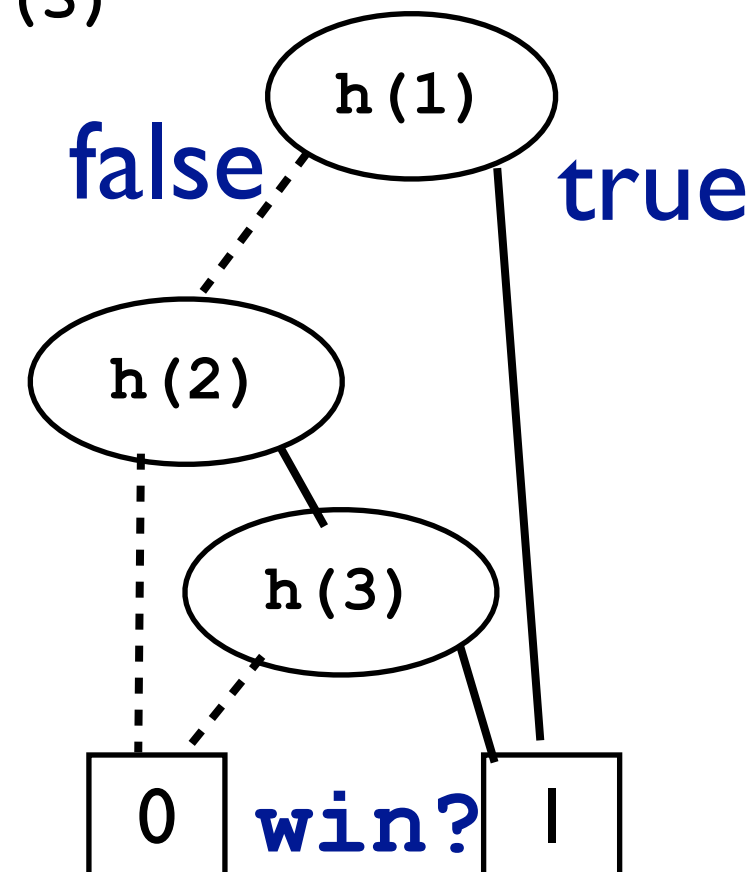
win

Find all proofs of query

Binary Decision
Diagram (BDD)

calculate marginal by
dynamic programming

heads(1)
heads(2) & heads(3)



Initial Approach

(ProbLogI & others)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),heads(3).
```

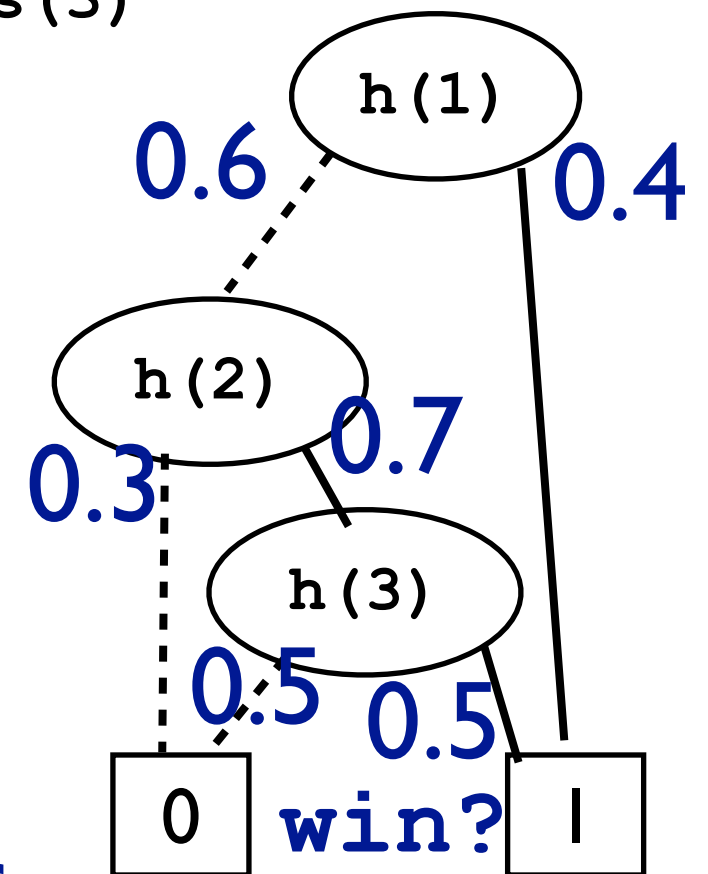
win

Find all proofs of query

Binary Decision
Diagram (BDD)

calculate marginal by
dynamic programming

heads(1)
heads(2) & heads(3)



$P(\text{win}) =$
probability of
reaching 1-leaf

Answering Questions

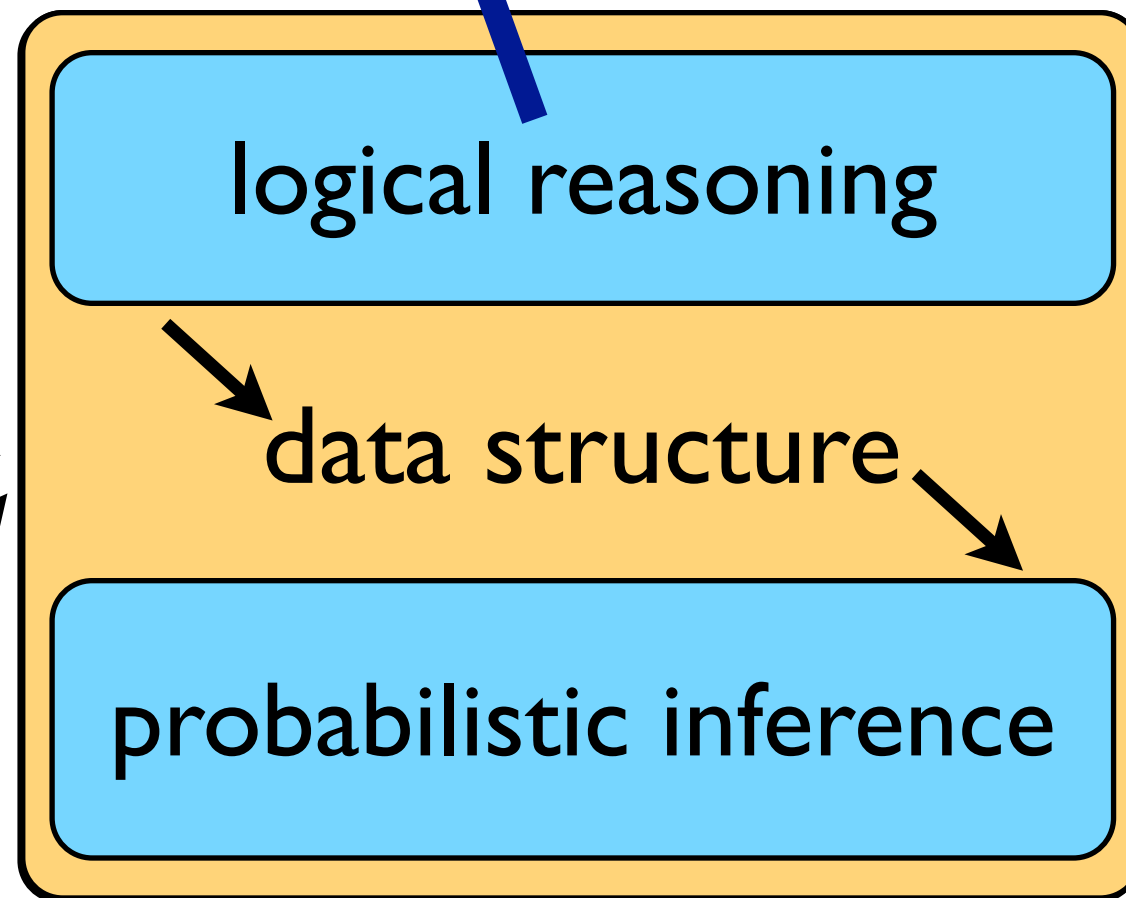
1. using proofs
2. using models

Given:

program

queries

evidence



Find:

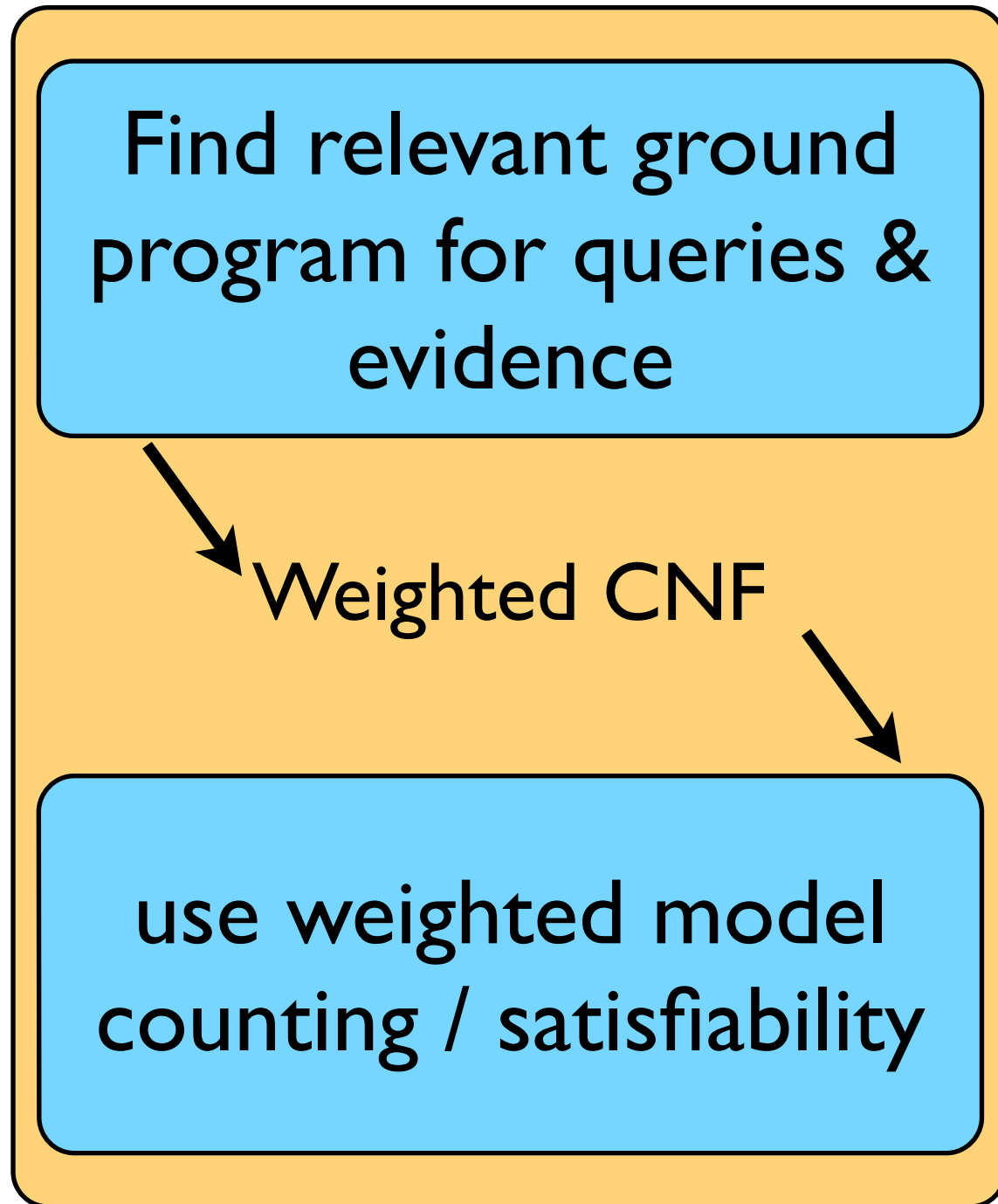
marginal
probabilities

conditional
probabilities

MPE state

Current Approach

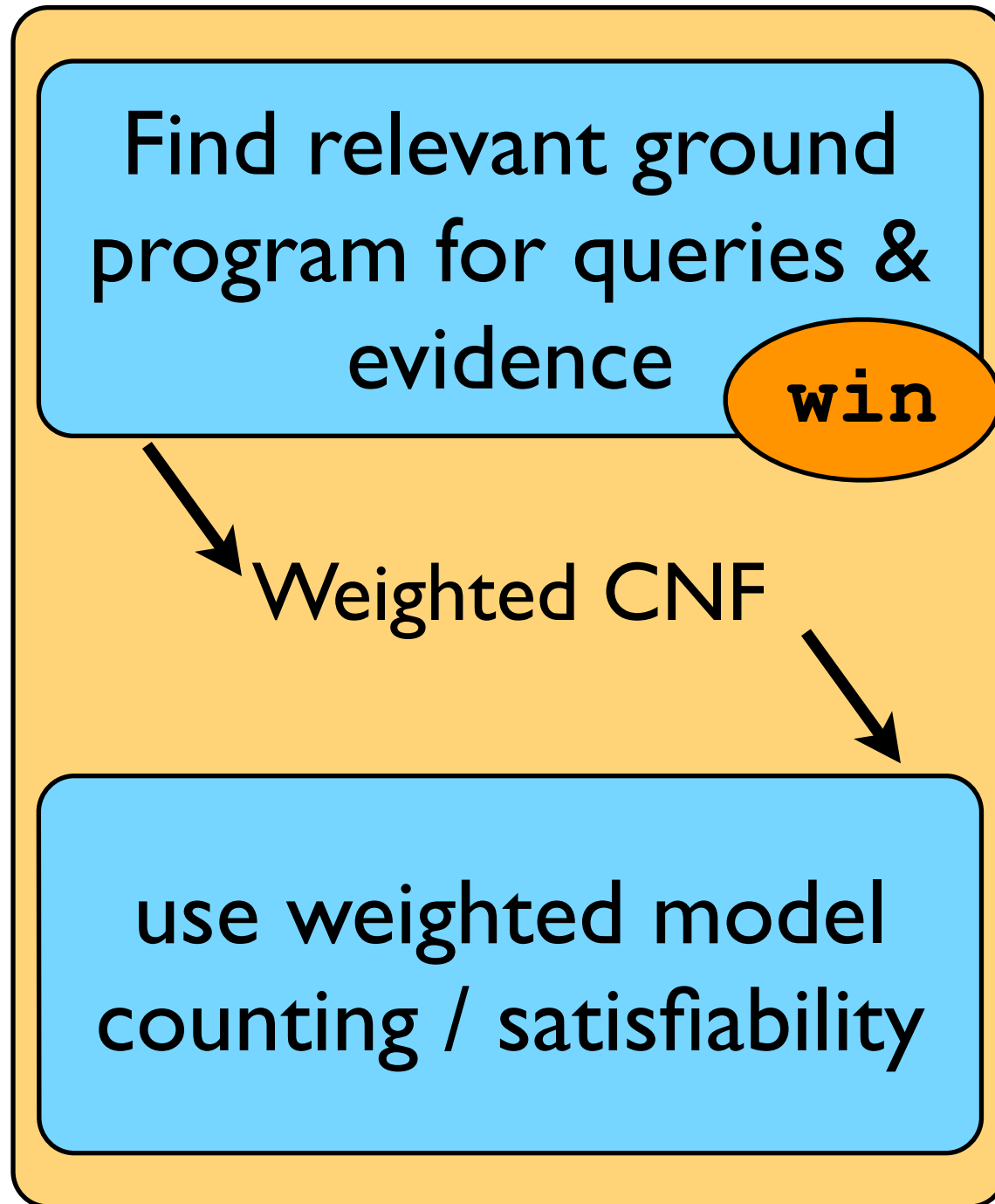
(ProbLog2)



Current Approach

(ProbLog2)

```
0.4::heads(1) .  
0.7::heads(2) .  
0.5::heads(3) .  
win :- heads(1) .  
win :- heads(2) ,  
           heads(3) .
```



Current Approach

(ProbLog2)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),  
        heads(3).
```

Find relevant ground
program for queries &
evidence

win

Weighted CNF

use weighted model
counting / satisfiability

```
win :- heads(1).  
win :- heads(2), heads(3).
```

Current Approach

(ProbLog2)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),  
        heads(3).
```

Find relevant ground
program for queries &
evidence

win

Weighted CNF

use weighted model
counting / satisfiability

```
win :- heads(1).  
win :- heads(2), heads(3).
```

↓
 $\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$

Current Approach

(ProbLog2)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),  
      heads(3).
```

Find relevant ground
program for queries &
evidence

win

Weighted CNF

use weighted model
counting / satisfiability

```
win :- heads(1).  
win :- heads(2), heads(3).
```

$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$

$(\neg \text{win} \vee h(1) \vee h(2))$
 $\wedge (\neg \text{win} \vee h(1) \vee h(3))$
 $\wedge (\text{win} \vee \neg h(1))$
 $\wedge (\text{win} \vee \neg h(2) \vee \neg h(3))$

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
      heads(3).
```

Find relevant ground
program for queries &
evidence

win

Weighted CNF

use weighted model
counting / satisfiability

```
win :- heads(1).
win :- heads(2), heads(3).
```

$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$

$(\neg \text{win} \vee h(1) \vee h(2))$
 $\wedge (\neg \text{win} \vee h(1) \vee h(3))$
 $\wedge (\text{win} \vee \neg h(1))$
 $\wedge (\text{win} \vee \neg h(2) \vee \neg h(3))$

$h(1) \rightarrow 0.4$	$h(2) \rightarrow 0.7$	$h(3) \rightarrow 0.5$
$\neg h(1) \rightarrow 0.6$	$\neg h(2) \rightarrow 0.3$	$\neg h(3) \rightarrow 0.5$

Current Approach

(ProbLog2)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),  
      heads(3).
```

Find relevant ground
program for queries &
evidence

win

Weighted CNF

use weighted model
counting / satisfiability

```
win :- heads(1).  
win :- heads(2), heads(3).
```

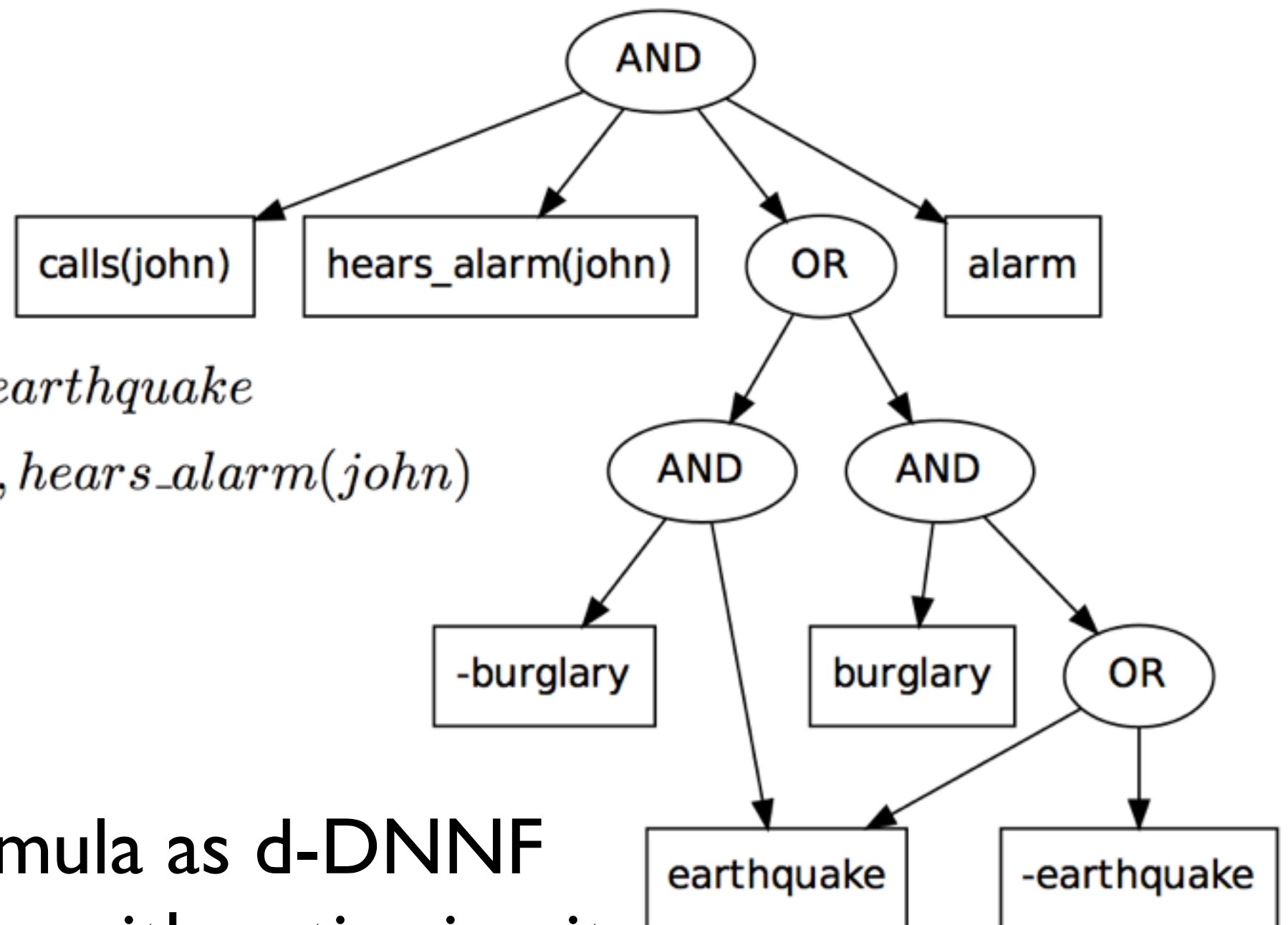
$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$

$(\neg \text{win} \vee h(1) \vee h(2))$
 $\wedge (\neg \text{win} \vee h(1) \vee h(3))$
 $\wedge (\text{win} \vee \neg h(1))$
 $\wedge (\text{win} \vee \neg h(2) \vee \neg h(3))$

use
standard
solver

$h(1) \rightarrow 0.4$	$h(2) \rightarrow 0.7$	$h(3) \rightarrow 0.5$
$\neg h(1) \rightarrow 0.6$	$\neg h(2) \rightarrow 0.3$	$\neg h(3) \rightarrow 0.5$

WMC using d-DNNFs



$alarm \leftrightarrow burglary \vee earthquake$

$calls(john) \leftrightarrow alarm, hears_alarm(john)$

$calls(john)$

1. represent formula as d-DNNF
2. transform into arithmetic circuit
3. evaluate bottom-up

Approximate Inference

- Lower and upper bounds

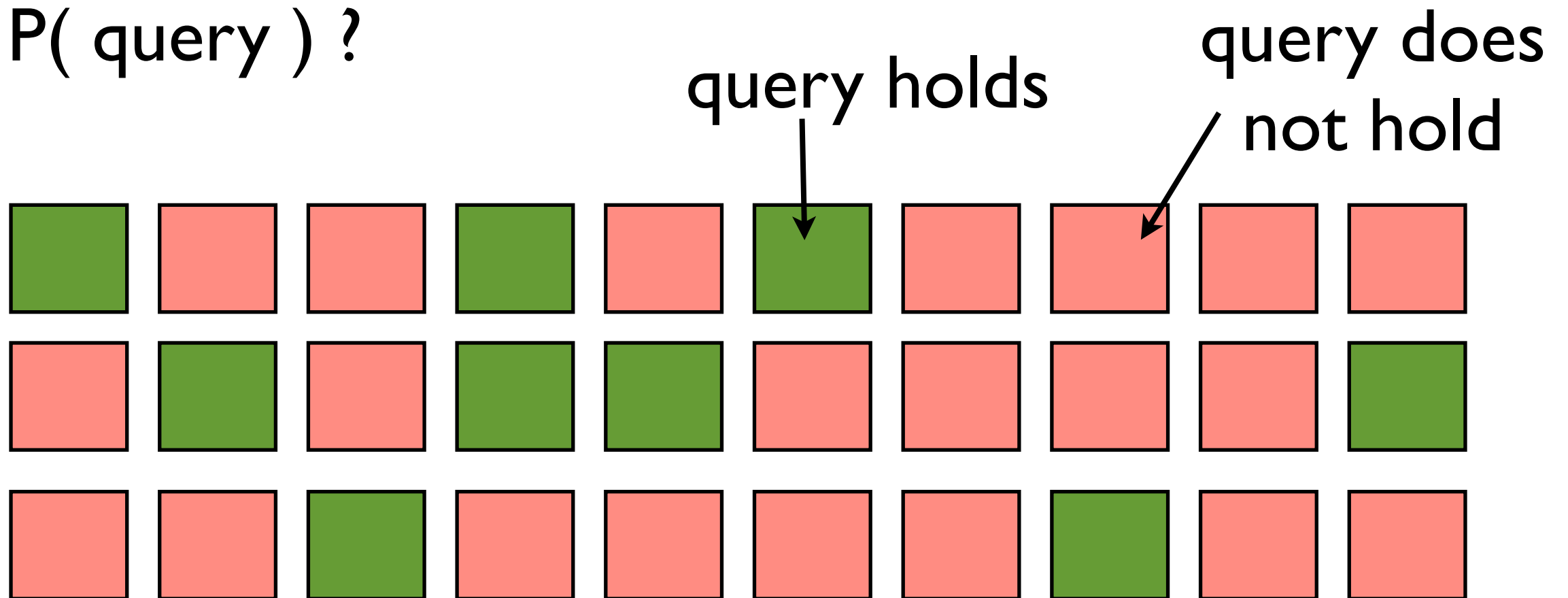
$$\phi_L \models \phi \models \phi_U$$

$$P(\phi_L) \leq P(\phi) \leq P(\phi_U)$$

- Sampling

Sampling

- $P(\text{query})$?



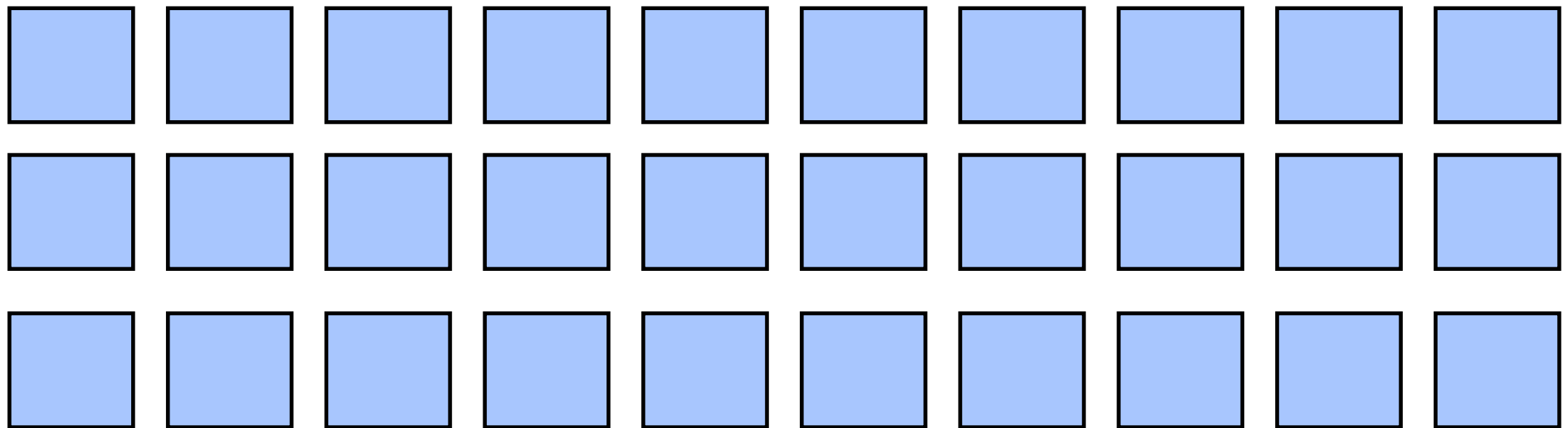
$$P(\text{query}) \approx \frac{\# \text{ query holds}}{\# \text{ worlds sampled}}$$

Rejection Sampling

- $P(\text{query} \mid \text{evidence})$?

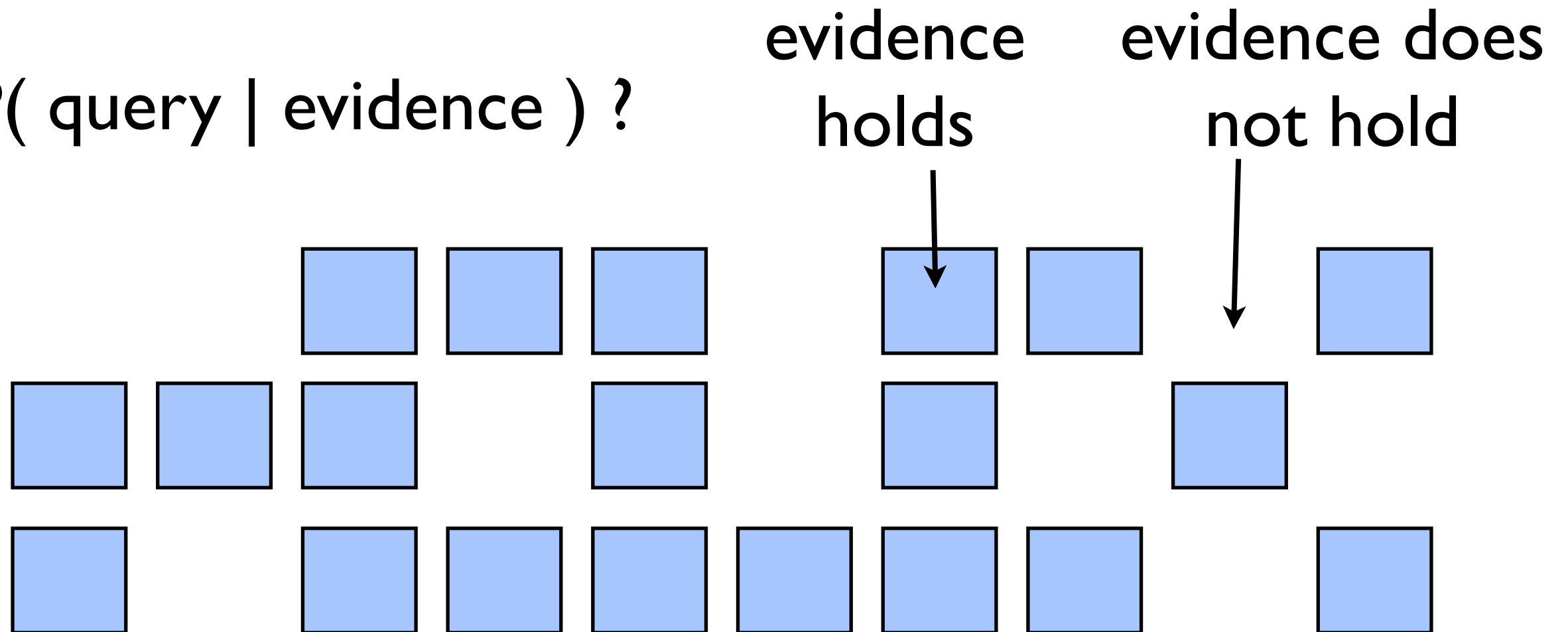
Rejection Sampling

- $P(\text{query} \mid \text{evidence})$?

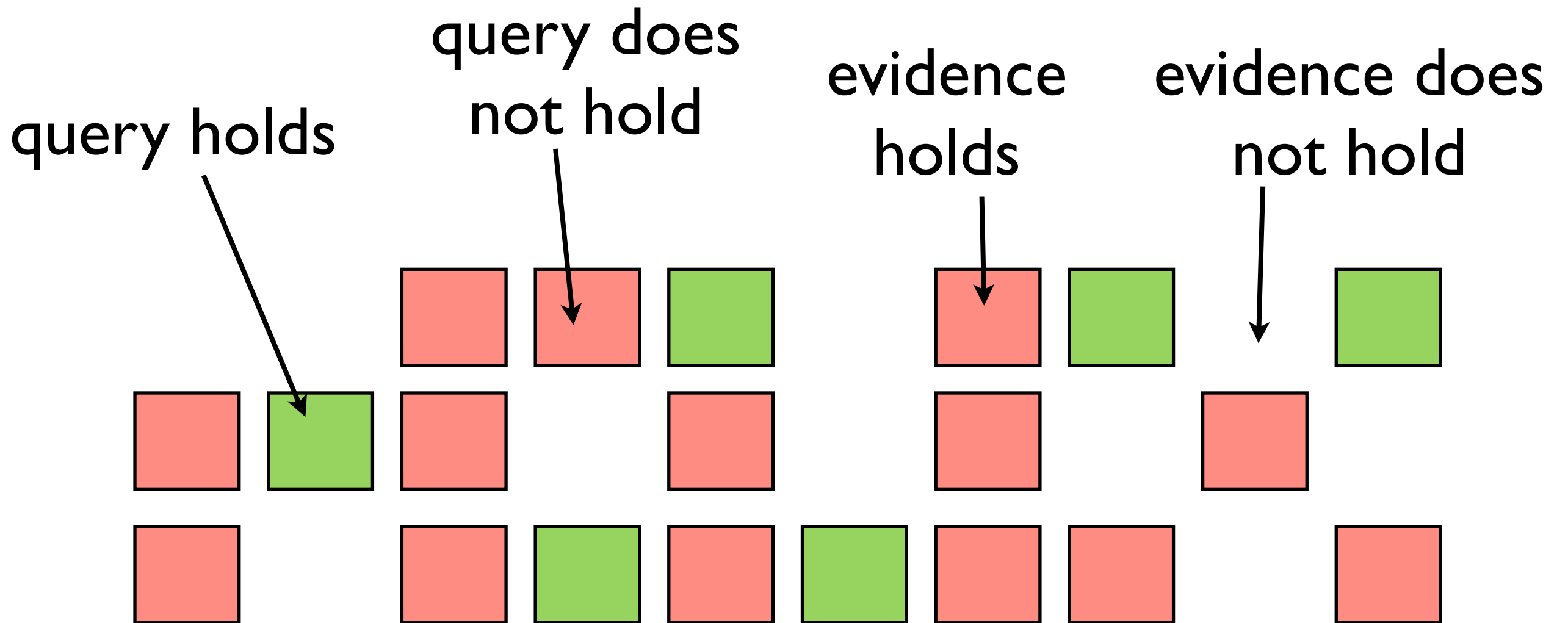


Rejection Sampling

- $P(\text{query} \mid \text{evidence})$?



Rejection Sampling



$$P(\text{query} \mid \text{evidence}) \approx \frac{\# \text{ query \& evidence holds}}{\# \text{ evidence holds}}$$

Markov Chain Monte Carlo (MCMC)

- Generate next sample by modifying current one
- Most common inference approach for PP languages such as Church, BLOG, ...
- Also considered for PRISM and ProbLog

Key challenges:

- how to propose next sample
- how to handle evidence

Roadmap

- Modeling (ProbLog and Church, another representative of PP)
- Inference
- Learning
- Dynamics and Decisions

... with some detours on the way

Parameter Learning

e.g., webpage classification model

for each *CLASS1*, *CLASS2* and each *WORD*

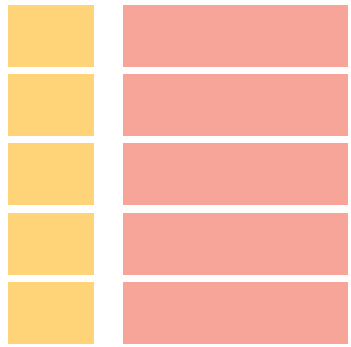
```
?? :: link_class(Source,Target,CLASS1,CLASS2).
```

```
?? :: word_class(WORD,CLASS).
```

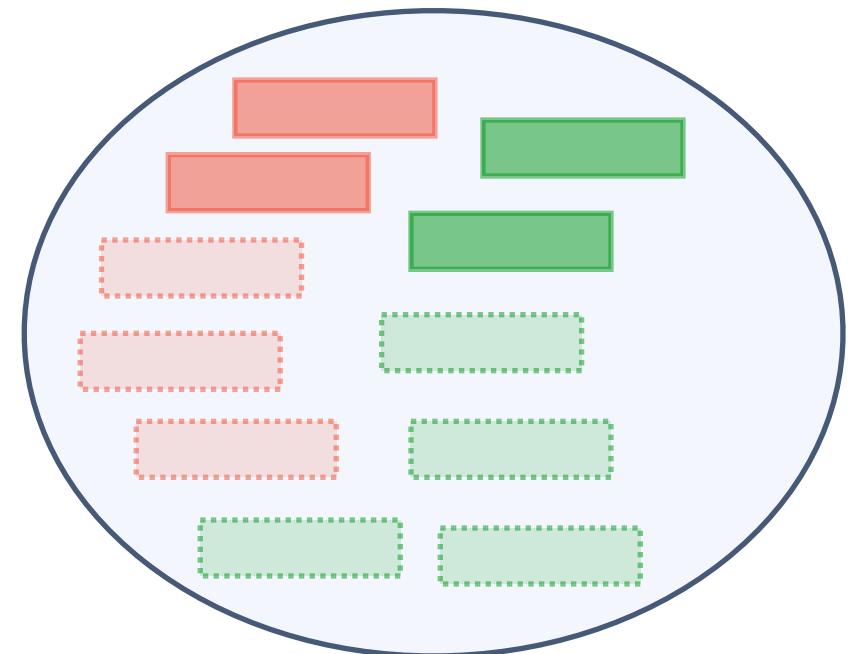
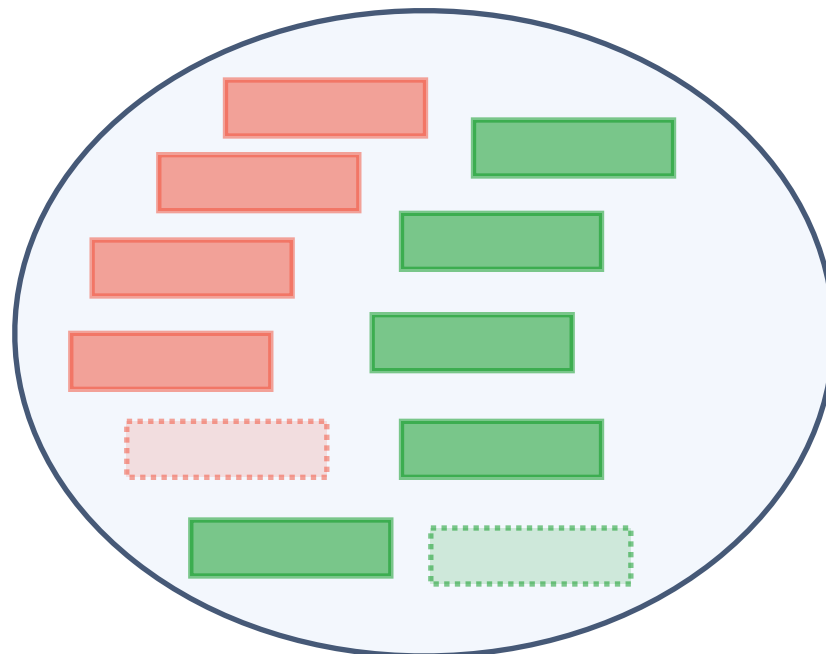
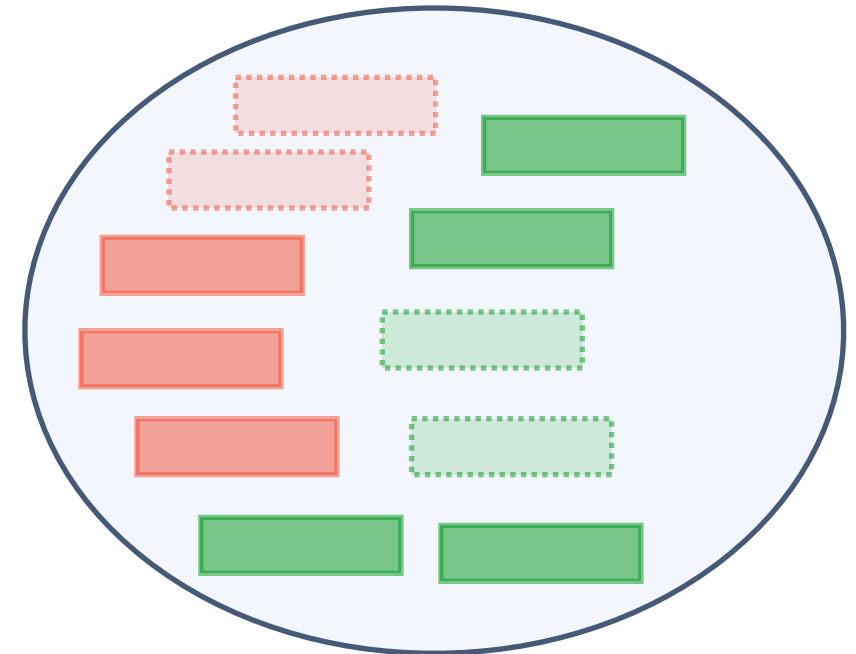
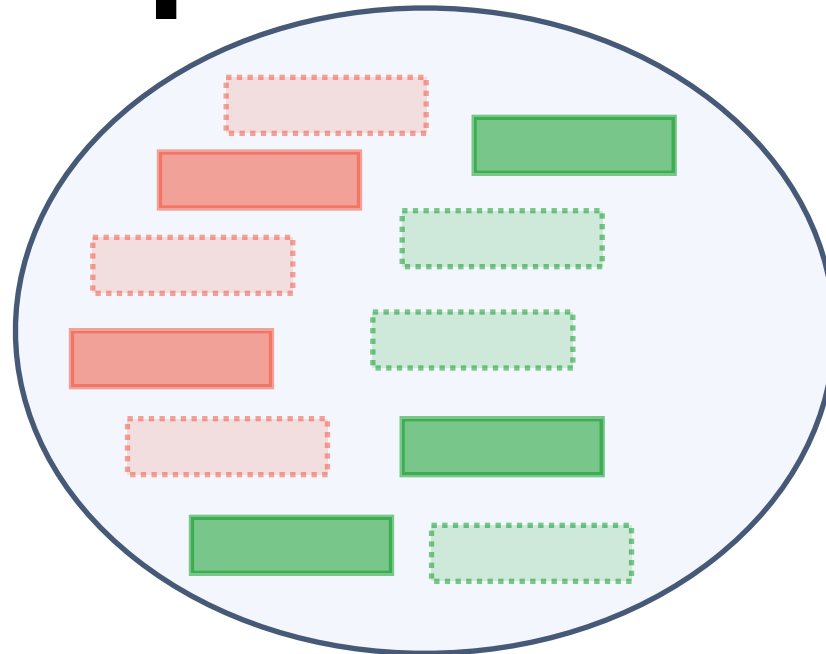
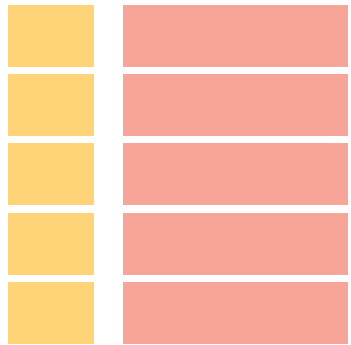
```
class(Page,C) :- has_word(Page,W), word_class(W,C).
```

```
class(Page,C) :- links_to(OtherPage,Page),  
    class(OtherPage,OtherClass),  
    link_class(OtherPage,Page,OtherClass,C).
```

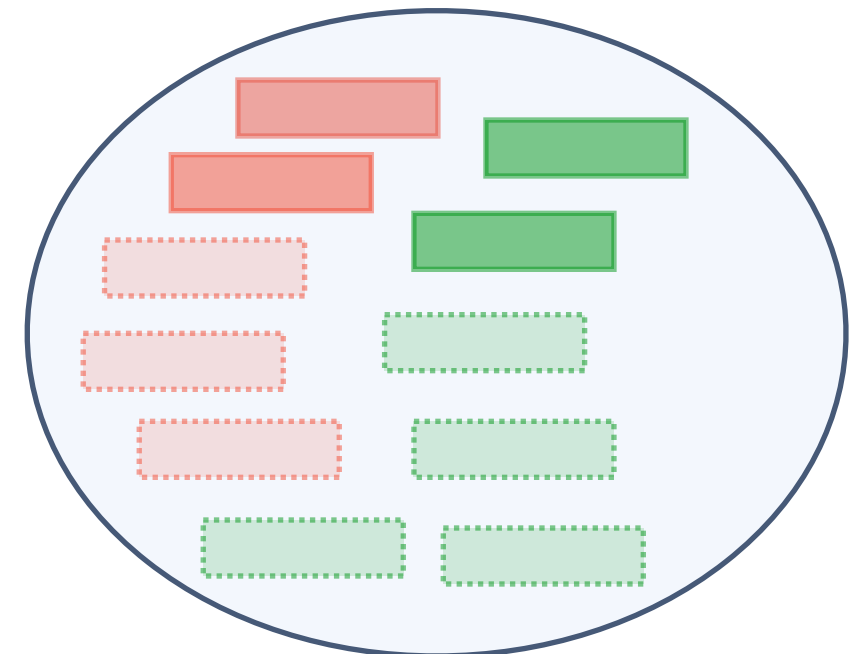
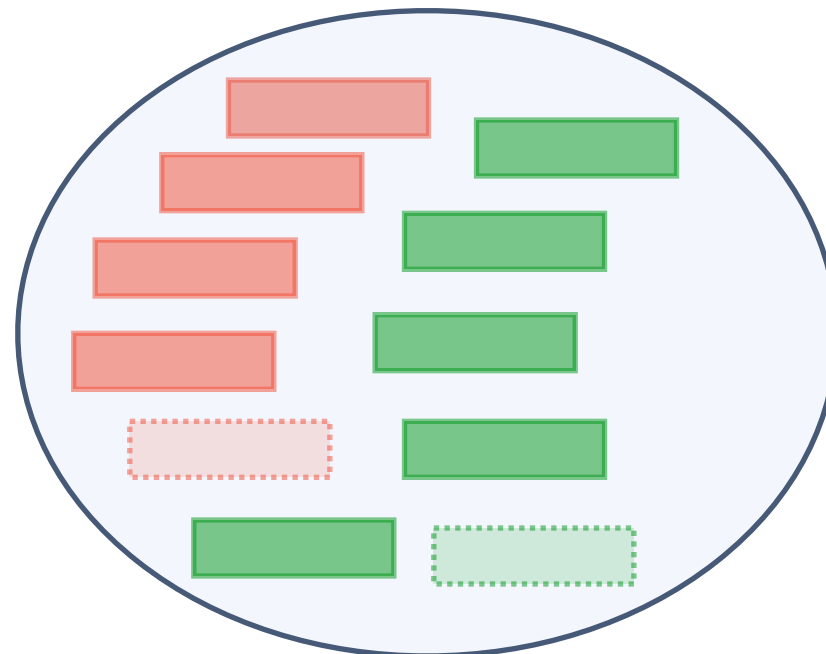
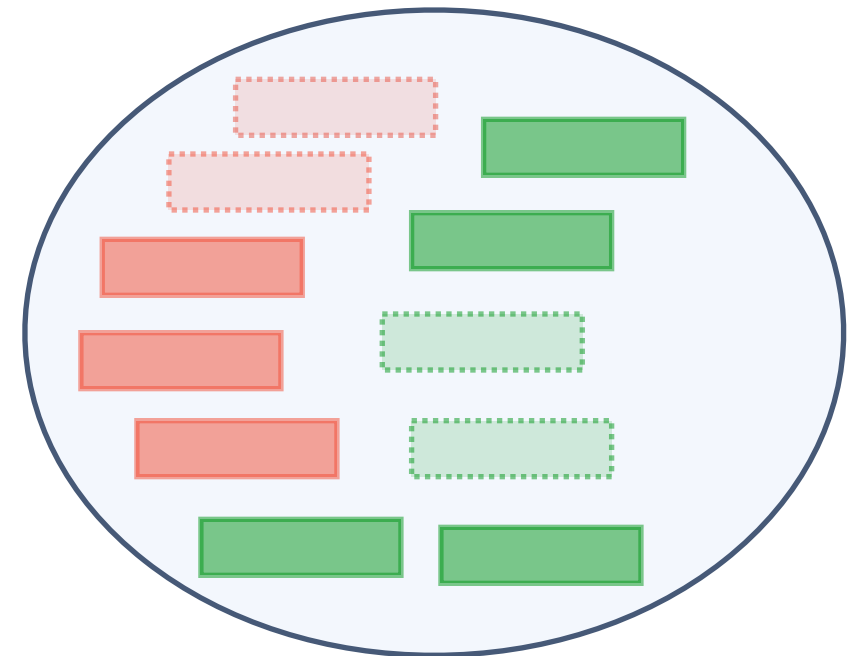
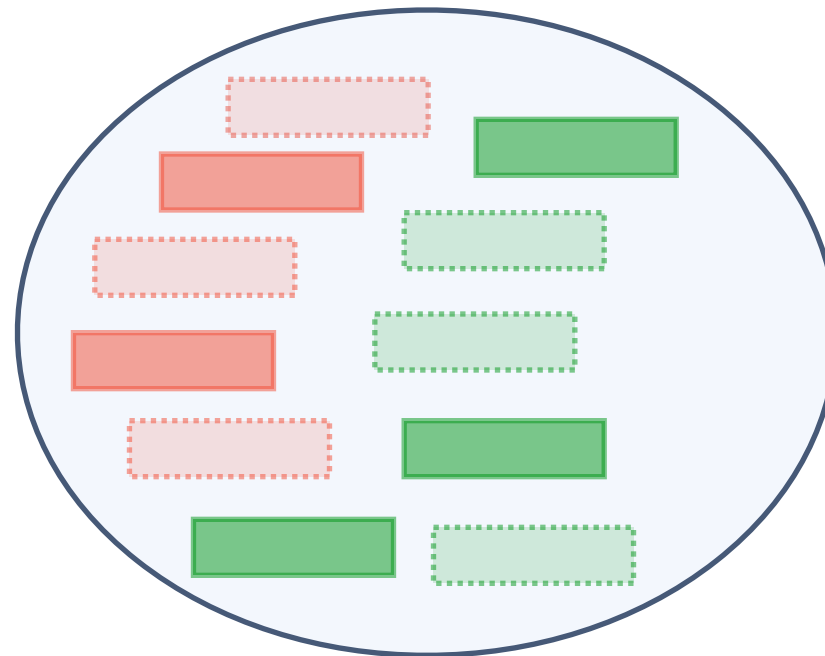
Sampling Interpretations



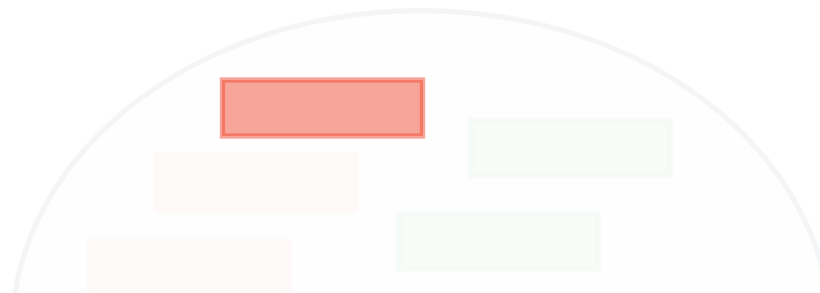
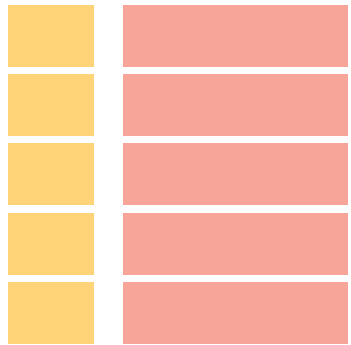
Sampling Interpretations



Parameter Estimation

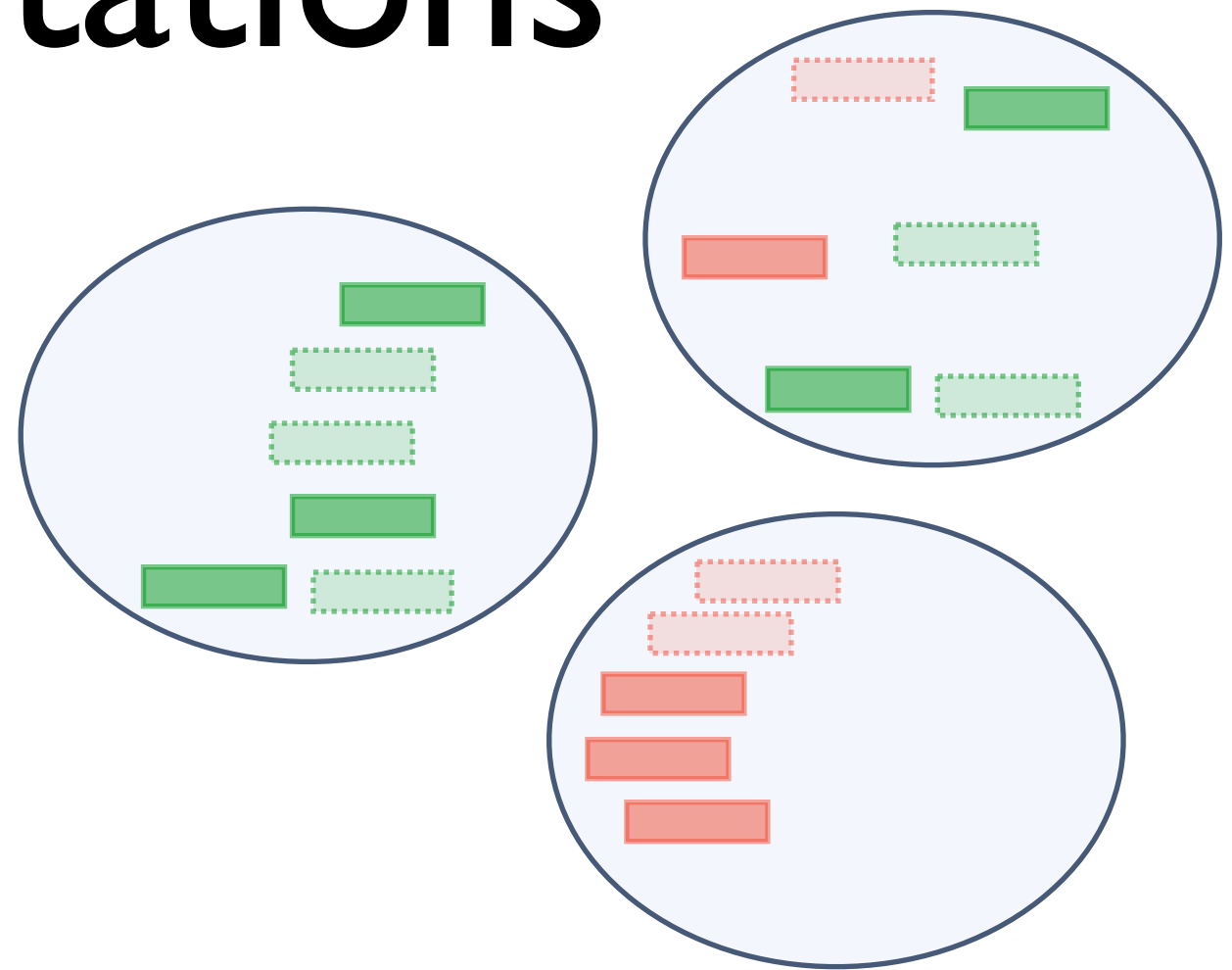


Parameter Estimation



$$p(\text{fact}) = \frac{\text{count}(\text{fact is true})}{\text{Number of interpretations}}$$

Learning from partial interpretations



- Not all facts observed
- Soft-EM
- use **expected count** instead of **count**
- $P(Q | E)$ -- conditional queries !

Bayesian Parameter Learning

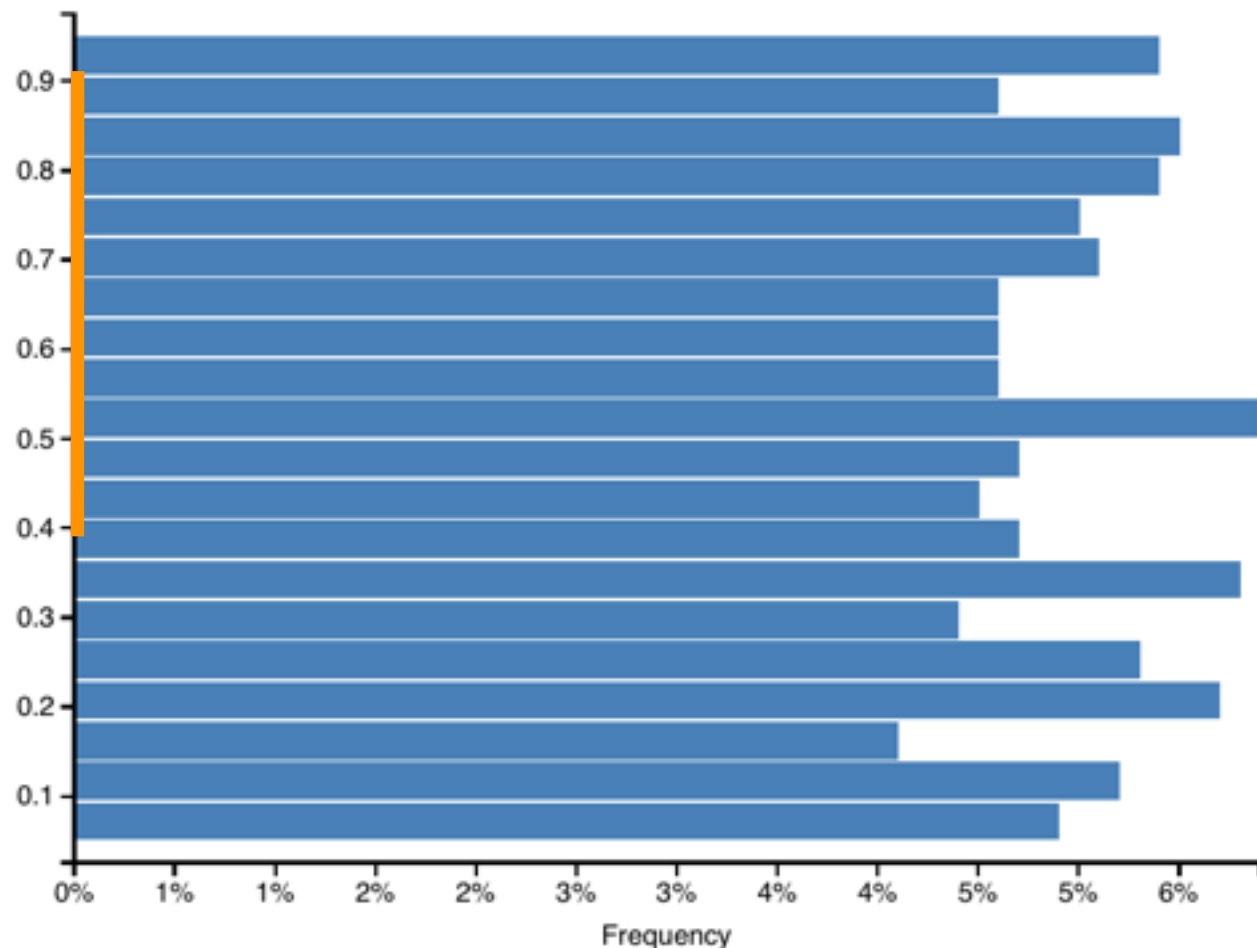
- Learning as inference (e.g., Church)
- Prior distributions for parameters
- Given data, find most likely parameter values

Example

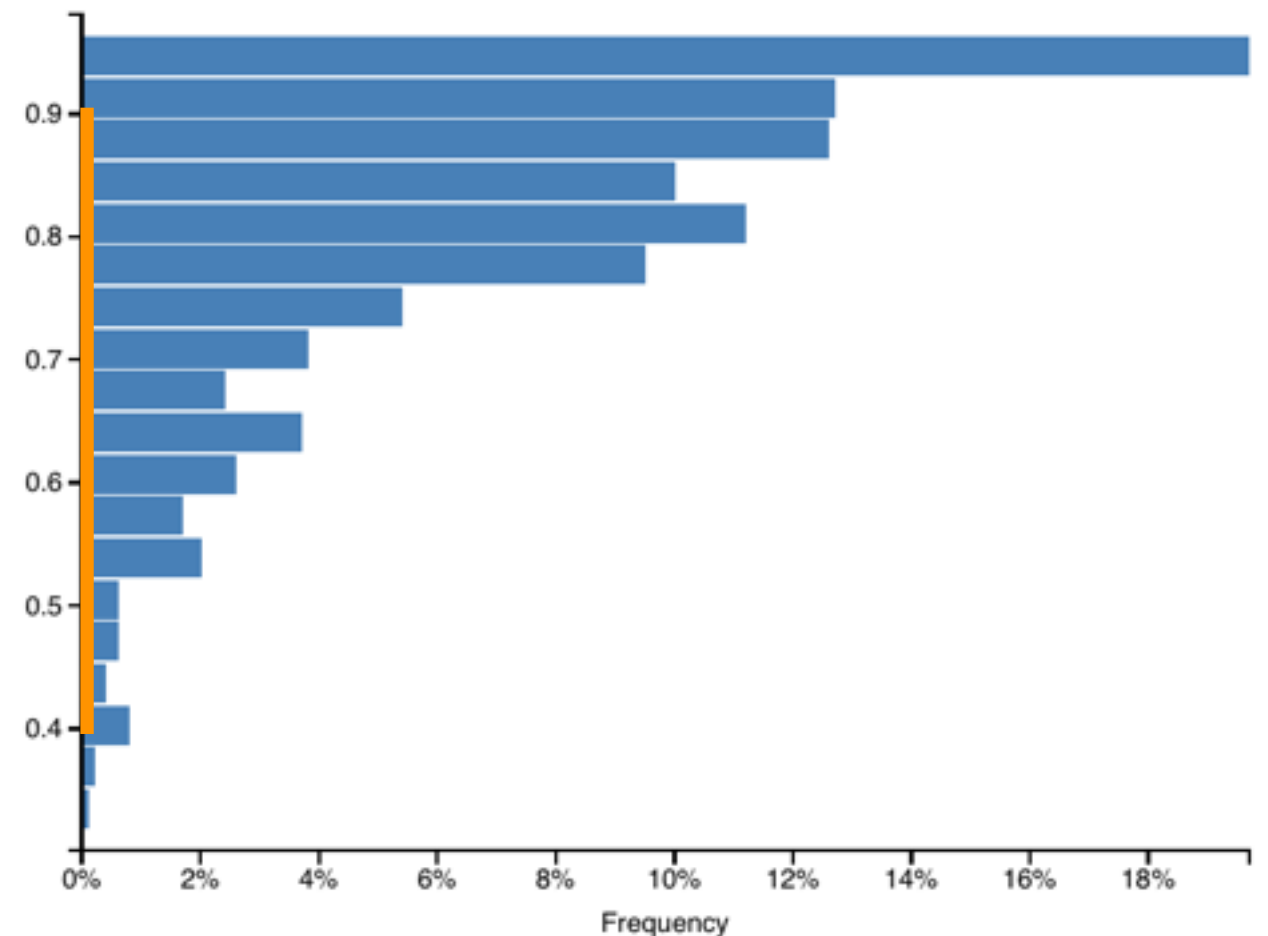
[from probmods.org]

- Flipping a coin with unknown weight
- Prior: uniform distribution on $[0, 1]$
- Observation: 5x heads in a row
- Sampling from Church model:

Coin weight, prior to observing data



Coin weight, conditioned on observed data



Rule learning — NELL

Table 5: Number of facts per predicate (NELL athlete dataset)

athletecoach(person,person)	18	athleteplaysforteam(person,team)	721
athleteplayssport(person,sport)	1921	teamplaysinleague(team,league)	1085
athleteplaysinleague(person,league)	872	athletealsoknownas(person,name)	17
coachesinleague(person,league)	93	coachesteam(person,team)	132
teamhomestadium(team,stadium)	198	teamplayssport(team,sport)	359
athleteplayssportsteamposition(person,position)	255	athletehomestadium(person,stadium)	187
athlete(person)	1909	attraction(stadium)	2
coach(person)	624	female(person)	2
male(person)	7	hobby(sport)	5
organization(league)	1	person(person)	2
personafrika(person)	1	personasia(person)	4
personaaustralia(person)	22	personcanada(person)	1
personeurope(person)	1	personmexico(person)	108
personus(person)	6	sport(sport)	36
sportsleague(league)	18	sportsteam(team)	1330
sportsteamposition(position)	22	stadiumoreventvenue(stadium)	171

Adaptation of standard rule learning and
inductive logic programming setting
[De Raedt et al IJCAI 15]

Experiments

Table 4: Precision for different experimental setups and parameters ($A: m = 1, p = 0.99, B: m = 1000, p = 0.90$).

Setting train/test/rule	athleteplaysforteam		athleteplayssport		teamplaysinleague		athleteplaysinleague		teamplaysagainstteam	
	A	B	A	B	A	B	A	B	A	B
1: det/det/det	74.00	69.36	94.14	93.47	96.29	82.15	80.95	74.14	73.40	73.86
2: det/prob/det	73.51	69.57	97.53	94.85	96.70	87.83	90.83	77.73	73.70	73.35
3: det/prob/prob	74.67	69.82	95.86	94.74	96.35	82.57	82.26	75.29	73.84	74.34
4: det/prob/prob	77.25	73.87	96.53	96.04	98.00	90.59	84.91	79.36	77.26	77.83
5: det/prob/prob	74.76	69.97	95.85	94.69	96.44	82.51	81.99	75.07	73.90	74.16
6: prob/prob/det	75.83	73.11	93.40	93.76	94.44	93.67	79.41	79.42	80.87	80.60
7: prob/prob/prob	78.31	73.72	95.62	95.10	98.84	91.86	96.94	79.49	85.78	81.81

Table 3: Learned relational rules for the different predicates (fold 1).

0.9375::athleteplaysforteam(A,B)	←	athleleedsportsteam(A,B).
0.9675::athleteplaysforteam(A,B)	←	athleleedsportsteam(A,V1), teamplaysagainstteam(B,V1).
0.9375::athleteplaysforteam(A,B)	←	athleteplayssport(A,V1), teamplayssport(B,V1).
0.5109::athleteplaysforteam(A,B)	←	athleteplaysinleague(A,V1), teamplaysinleague(B,V1).
0.9070::athleteplayssport(A,B)	←	athleleedsportsteam(A,V2), teamalsoknownas(V2,V1), teamplayssport(V1,B), teamplayssport(V2,B).
0.9070::athleteplayssport(A,B)	←	athleteplaysforteam(A,V2), teamalsoknownas(V2,V1), teamplayssport(V1,B), teamplayssport(V2,B), teamalsoknownas(V1,V2).
0.9070::athleteplayssport(A,B)	←	athleteplaysforteam(A,V1), teamplayssport(V1,B).
0.9286::athleteplaysinleague(A,B)	←	athleleedsportsteam(A,V1), teamplaysinleague(V1,B).
0.7868::athleteplaysinleague(A,B)	←	athleteplaysforteam(A,V2), teamalsoknownas(V2,V1), teamplaysinleague(V1,B).
0.9384::athleteplaysinleague(A,B)	←	athleteplayssport(A,V2), athleteplayssport(V1,V2), teamplaysinleague(V1,B).
0.9024::athleteplaysinleague(A,B)	←	athleteplaysforteam(A,V1), teamplaysinleague(V1,B).

Roadmap

- Modeling (ProbLog and Church, another representative of PP)
- Inference
- Learning
- Dynamics and Decisions

... with some detours on the way

World Dynamics

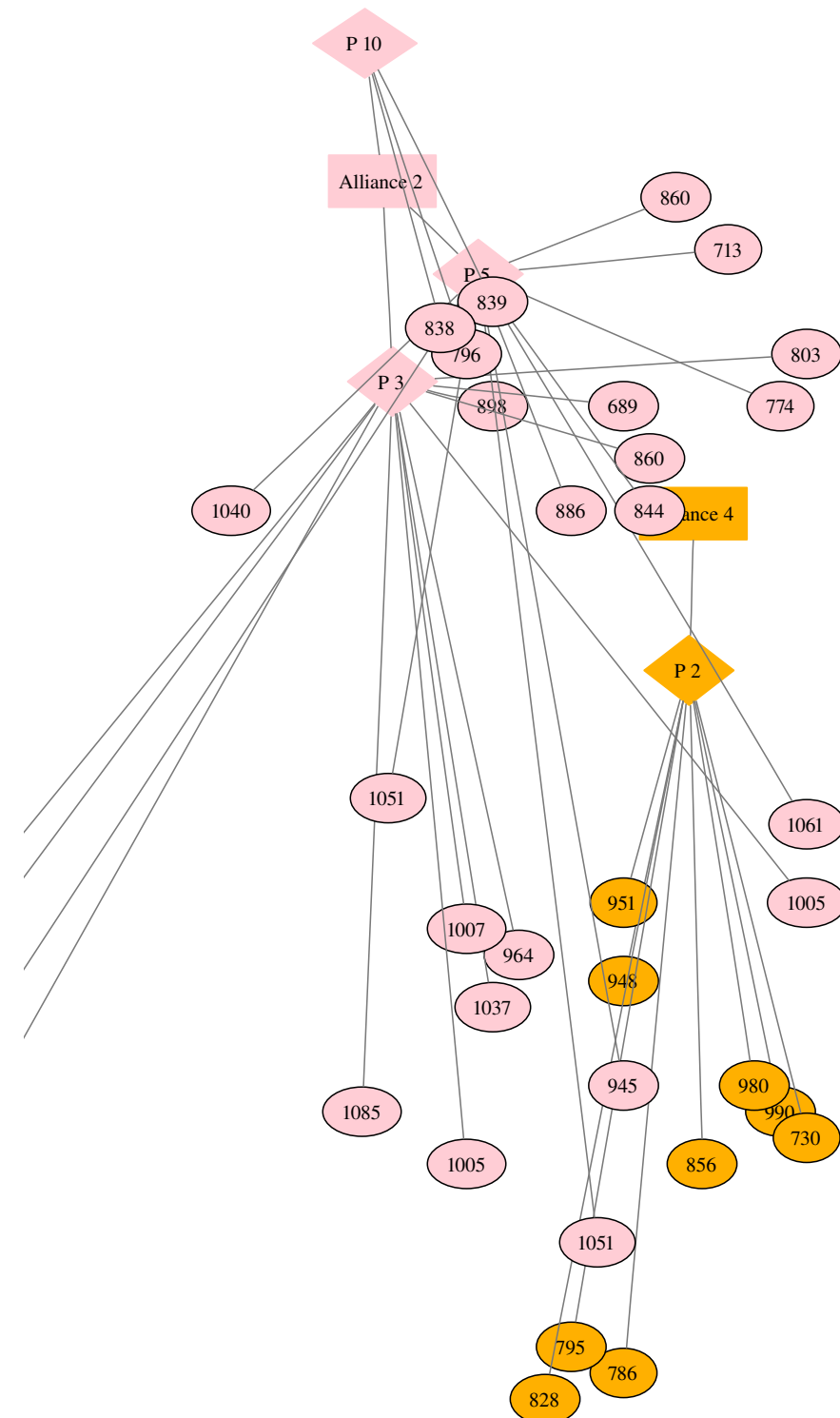
Fragment of world with

~10 alliances
~200 players
~600 cities

alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?

[Thon, Landwehr, De Raedt, ECML08]



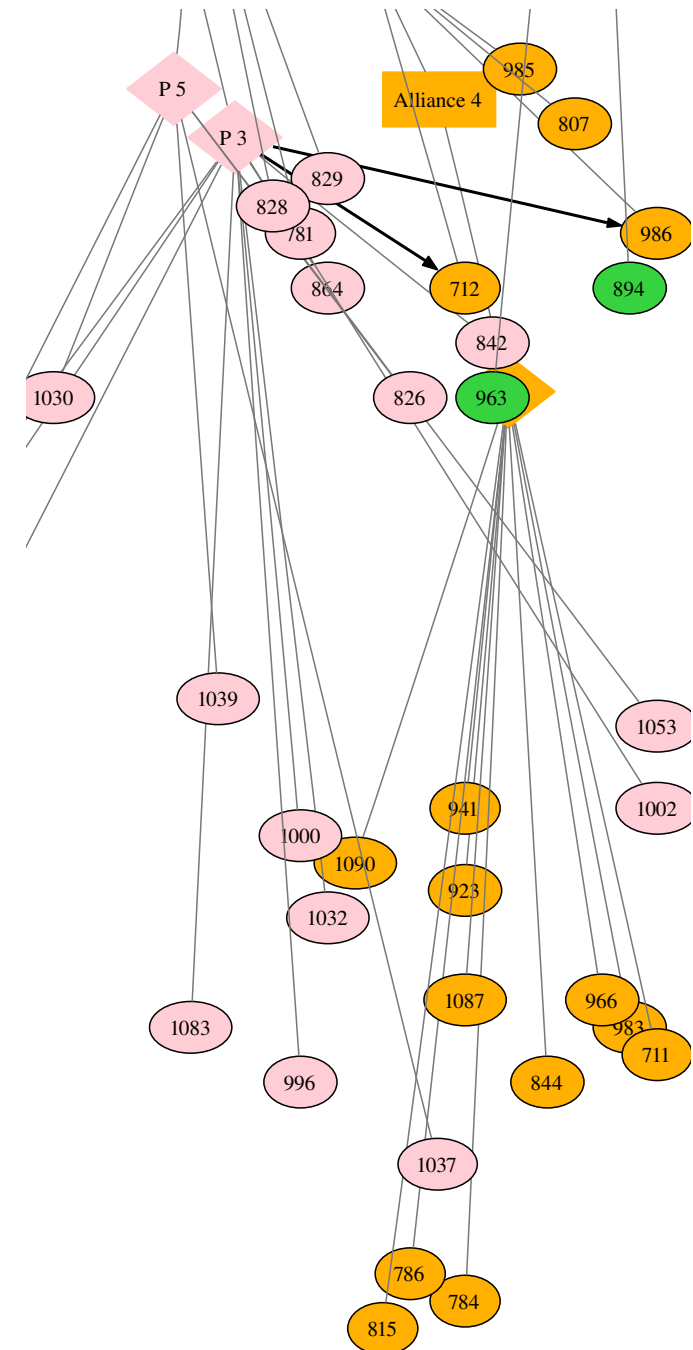
CPT-Rules

$$\frac{b_1, \dots, b_n}{\text{cause}} \rightarrow \frac{h_1 : p_1 \vee \dots \vee h_m : p_m}{\text{effect}}$$

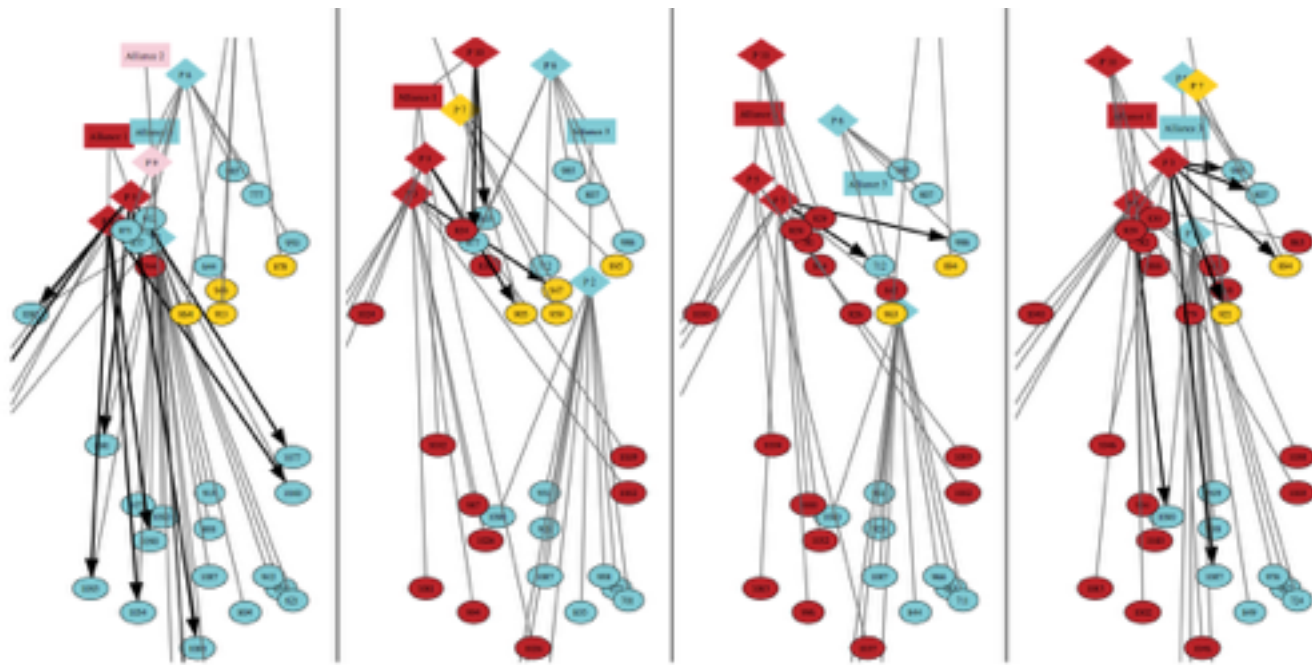
$city(C, Owner), city(C2, Attacker), close(C, C2) \rightarrow$
 $conquest(Attacker, C2) : p \vee nil : (1 - p)$

conquer a city which is close
 $P(conquest(), Time+5)$?
 learn parameters

Thon et al. MLJ I I

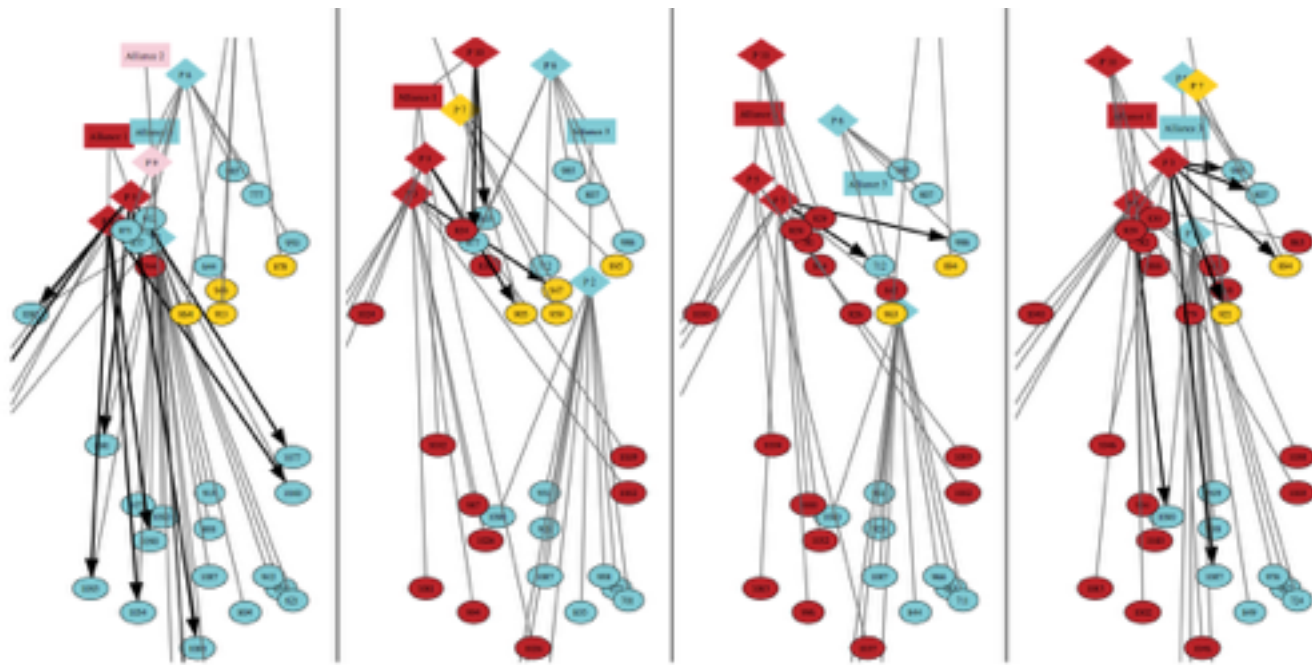


Causal Probabilistic Time-Logic (CPT-L)



how does the
world change
over time?

Causal Probabilistic Time-Logic (CPT-L)



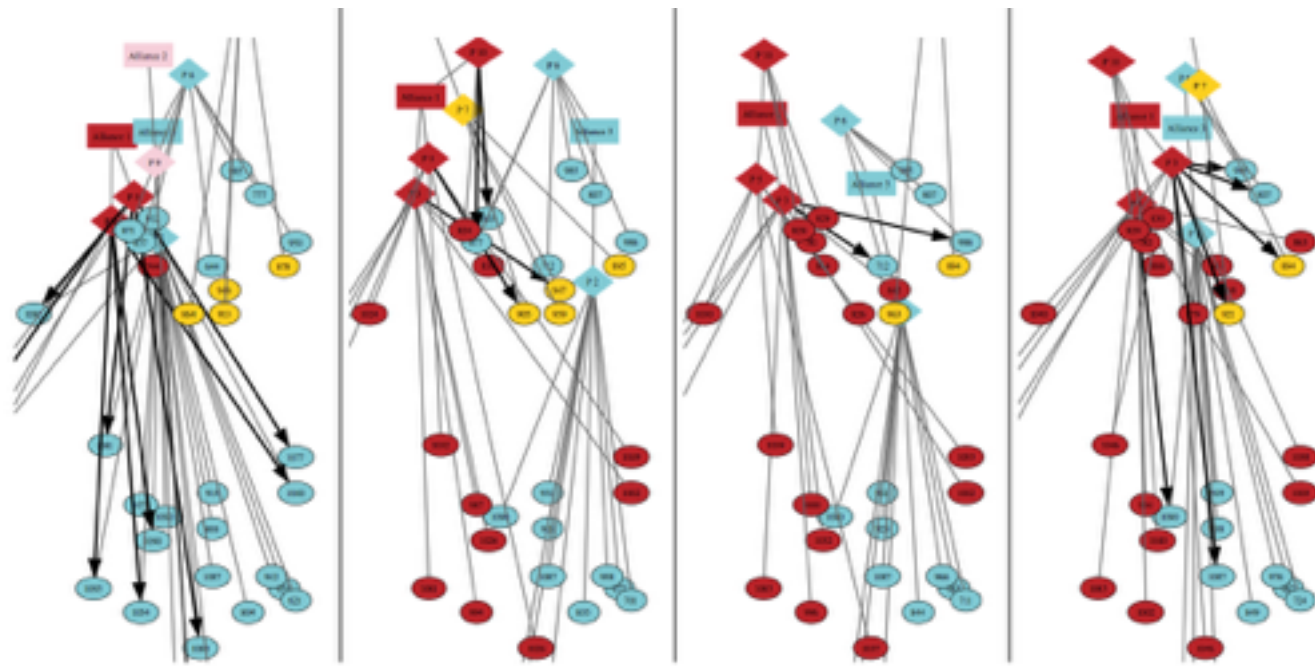
how does the
world change
over time?

```
0.4::conquest(Attacker,C) ; 0.6::nil :-
```

```
city(C,Owner) , city(C2,Attacker) , close(C,C2) .
```

if **cause** holds at time T

Causal Probabilistic Time-Logic (CPT-L)



how does the world change over time?

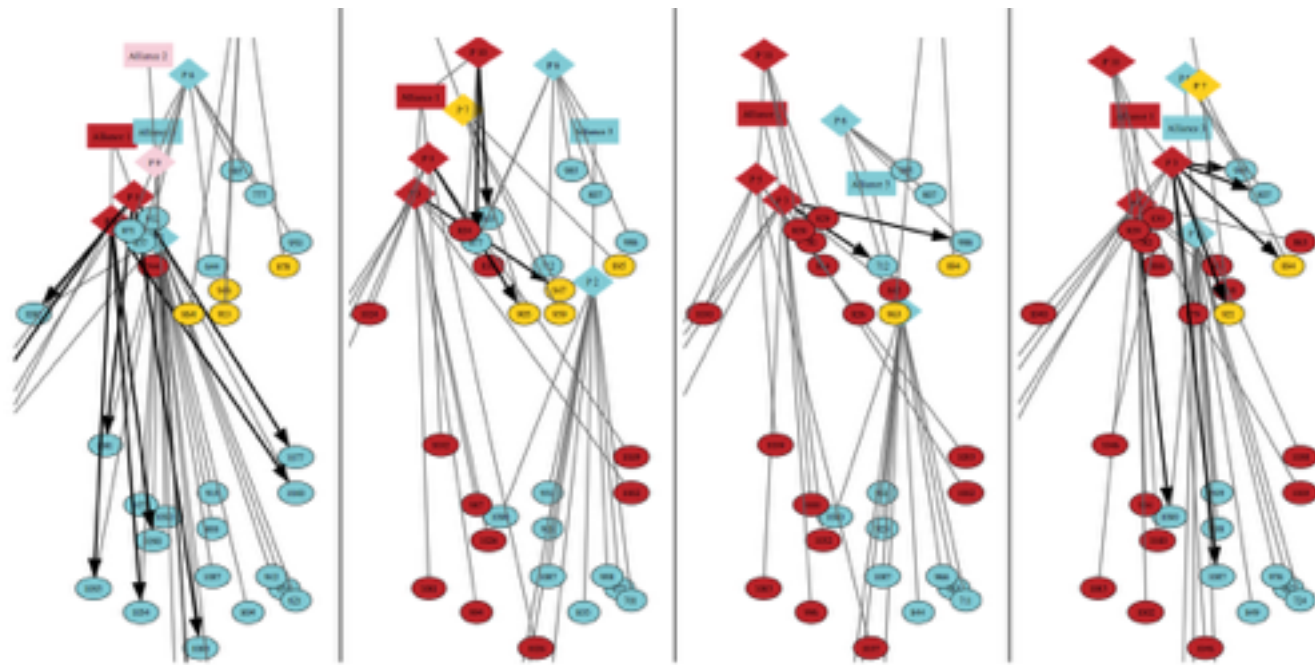
one of the **effects** holds at time $T+1$

```
0.4::conquest(Attacker,C) ; 0.6::nil :-
```

```
city(C,Owner) , city(C2,Attacker) , close(C,C2) .
```

if **cause** holds at time T

Causal Probabilistic Time-Logic (CPT-L)



how does the
world change
over time?

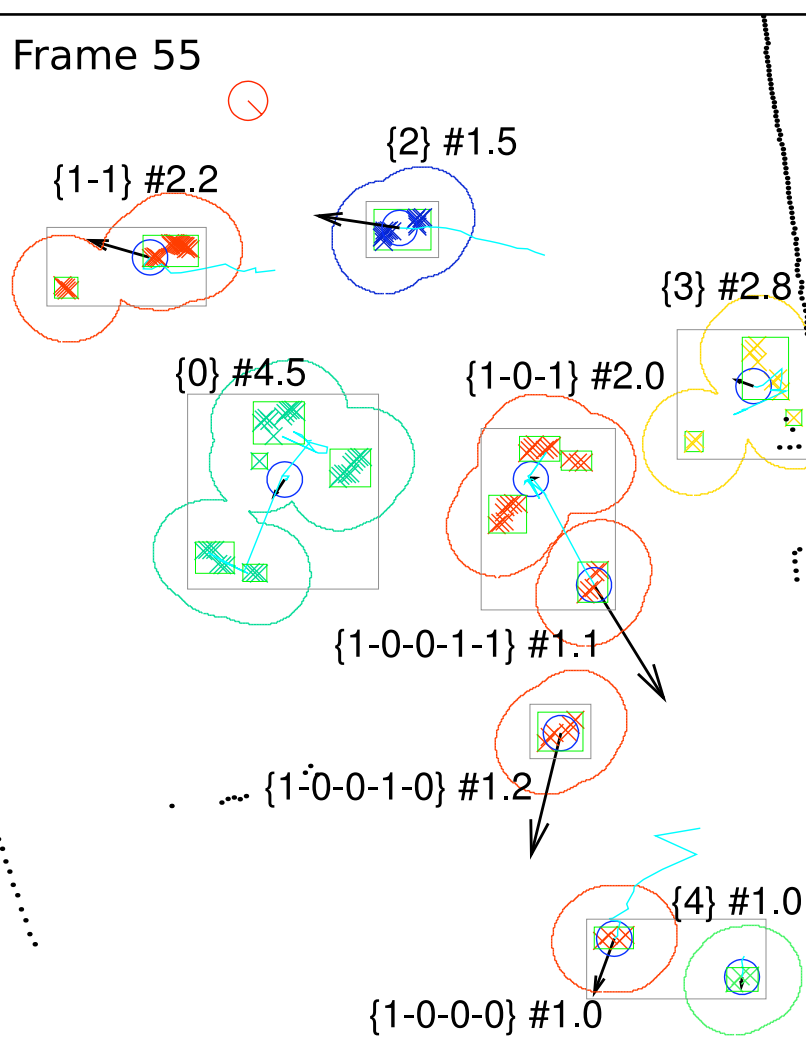
one of the **effects** holds at time $T+1$

```
0.4::conquest(Attacker,C) ; 0.6::nil :-
```

```
city(C,Owner) , city(C2,Attacker) , close(C,C2) .
```

if **cause** holds at time T

Analyzing Video Data



- Track people or objects over time? Even if temporarily hidden?

[Skarlatidis et al, TPLP 14;
Nitti et al, IROS 13, ICRA 14]

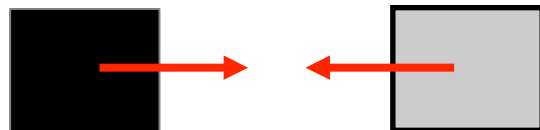
- Recognize activities?
- Infer object properties?

Magnetic scenario

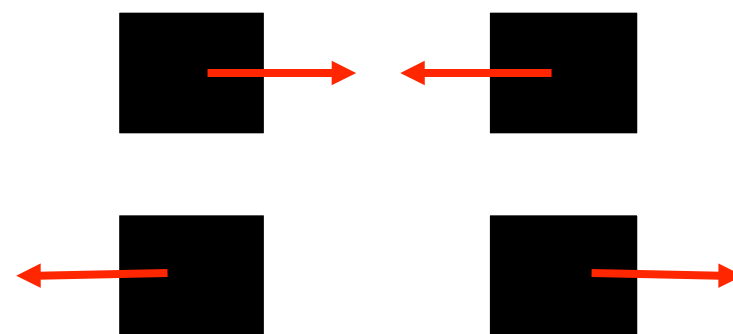
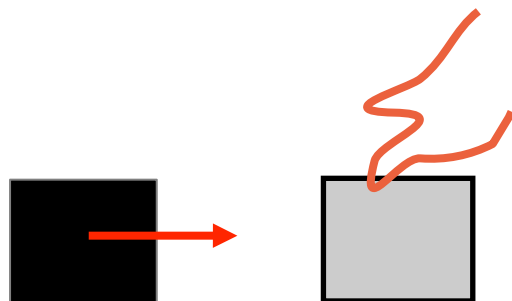
- 3 object types: magnetic, ferromagnetic, nonmagnetic

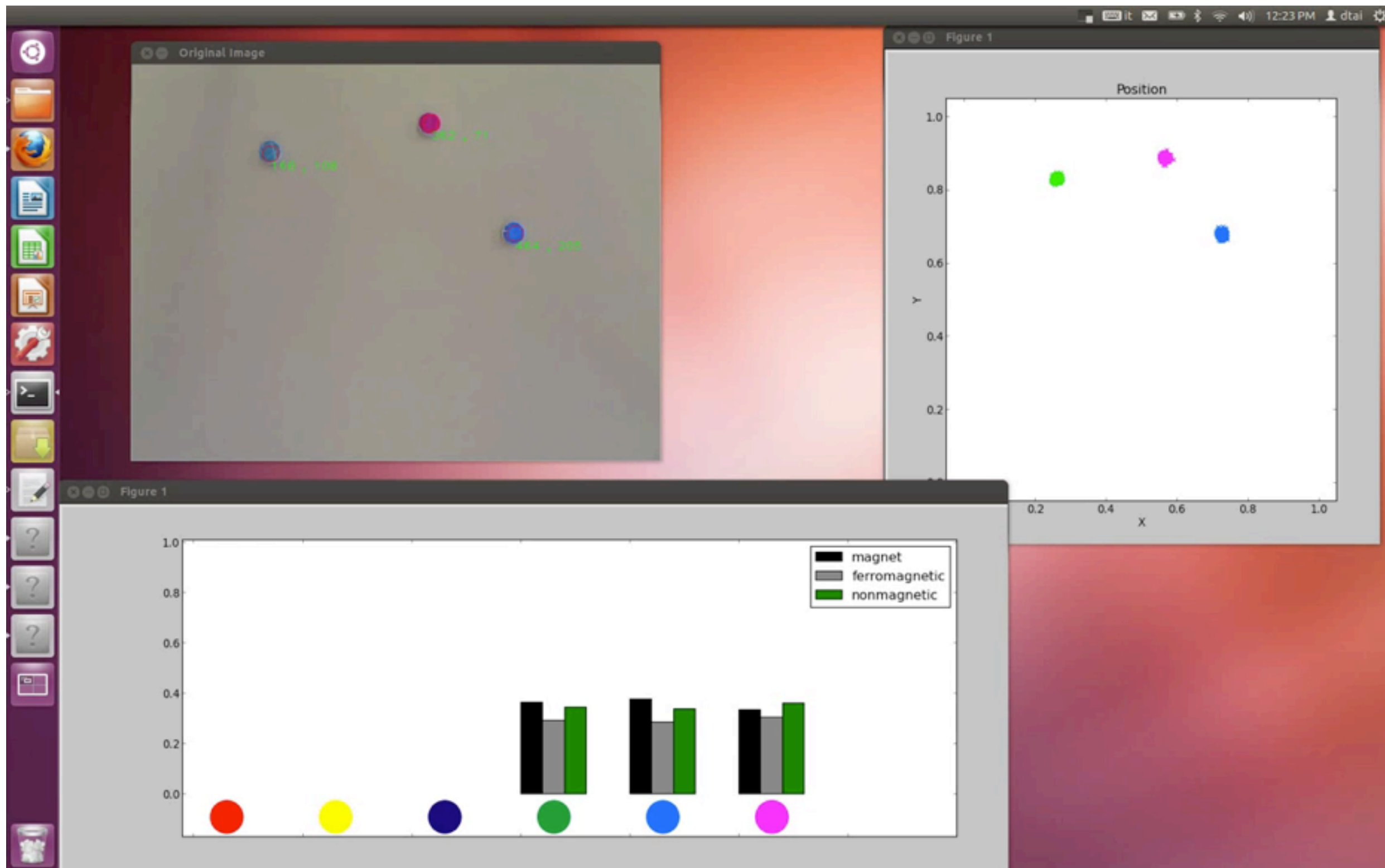


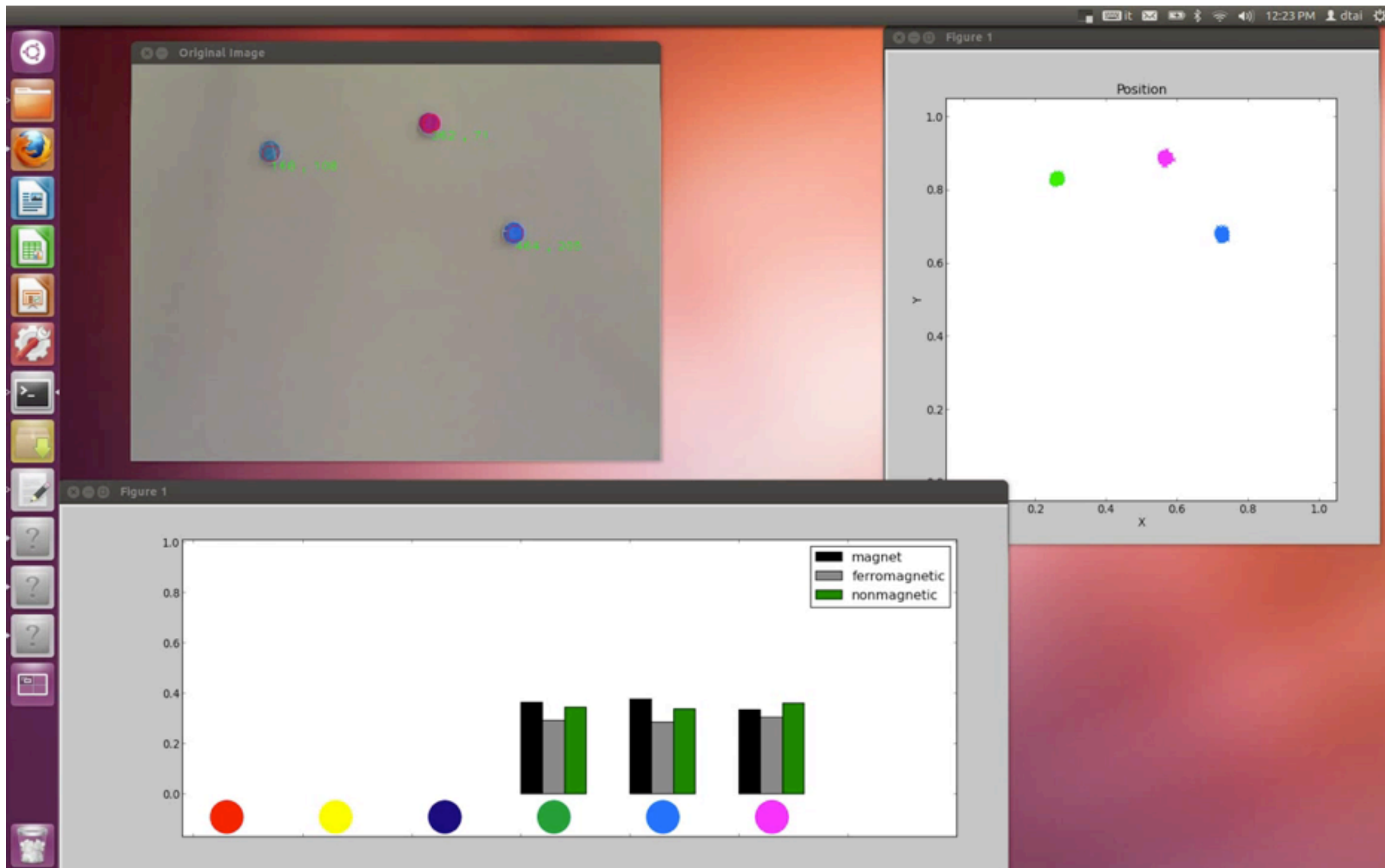
- Nonmagnetic objects do not interact
- A magnet and a ferromagnetic object attract each other



- Magnetic force that depends on the distance
- If an object is held magnetic force is compensated.







Magnetic scenario

- 3 object types: magnetic, ferromagnetic, nonmagnetic

$\text{type}(X)_t \sim \text{finite}([1/3:\text{magnet}, 1/3:\text{ferromagnetic}, 1/3:\text{nonmagnetic}]) \leftarrow \text{object}(X).$

- 2 magnets attract or repulse

$\text{interaction}(A,B)_t \sim \text{finite}([0.5:\text{attraction}, 0.5:\text{repulsion}]) \leftarrow \text{object}(A), \text{object}(B), A < B, \text{type}(A)_t = \text{magnet}, \text{type}(B)_t = \text{magnet}.$

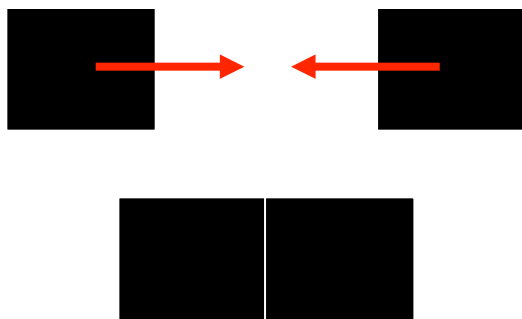
- Next position after attraction

$\text{pos}(A)_{t+1} \sim \text{gaussian}(\text{midpoint}(A,B)_t, \text{Cov}) \leftarrow$

$\text{near}(A,B)_t, \text{not}(\text{held}(A)), \text{not}(\text{held}(B)),$

$\text{interaction}(A,B)_t = \text{attr},$

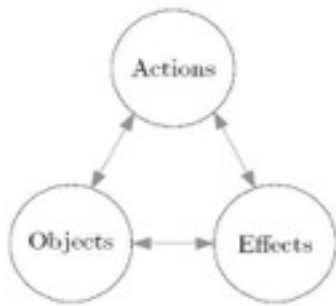
$c/\text{dist}(A,B)_t^2 > \text{friction}(A)_t.$



$\text{pos}(A)_{t+1} \sim \text{gaussian}(\text{pos}(A)_t, \text{Cov}) \leftarrow \text{not}(\text{attraction}(A,B)).$

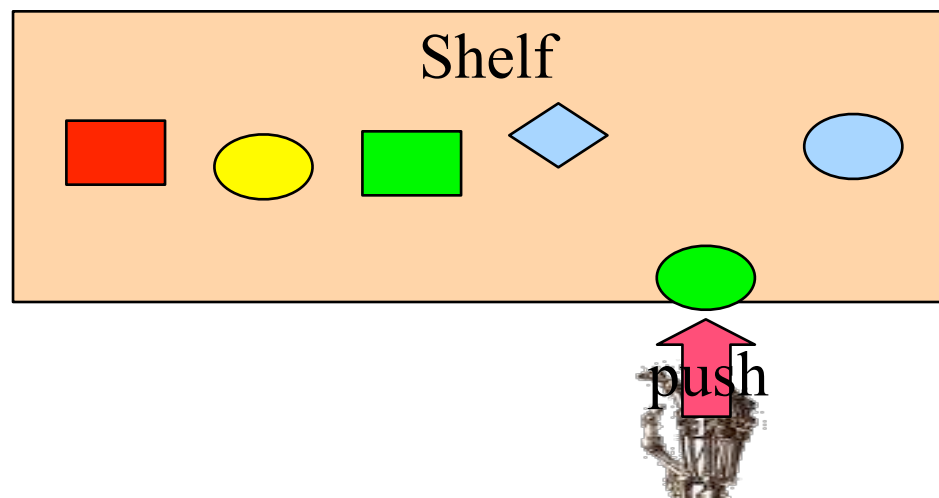
Learning relational affordances

Learn probabilistic model



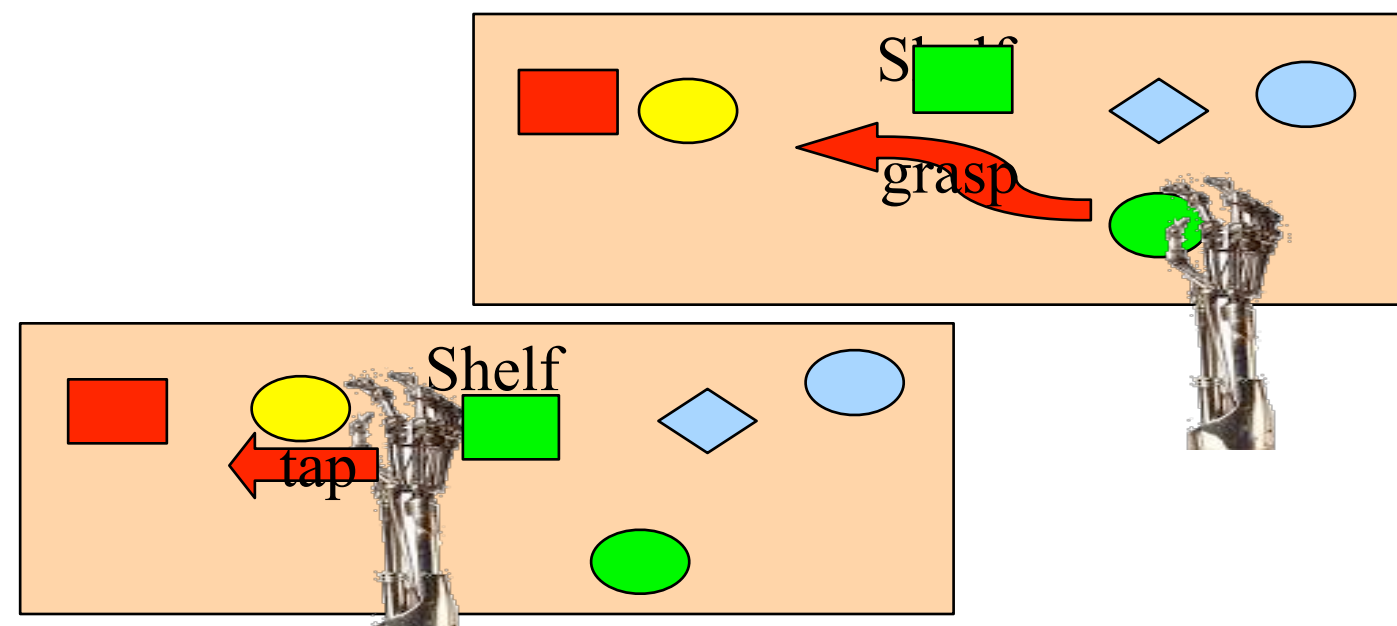
Inputs	Outputs	Function
(O, A)	E	Effect prediction
(O, E)	A	Action recognition/planning
(A, E)	O	Object recognition/selection

From two object interactions
Generalize to N



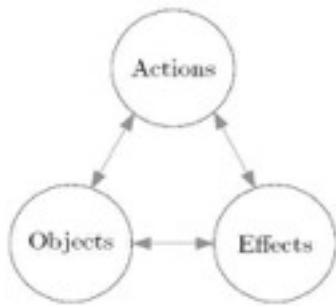
Learning relational
affordances
between
two objects
(learnt by experience)

Moldovan et al. ICRA 12, 13, 14, PhD 15



Learning relational affordances

Learn probabilistic model

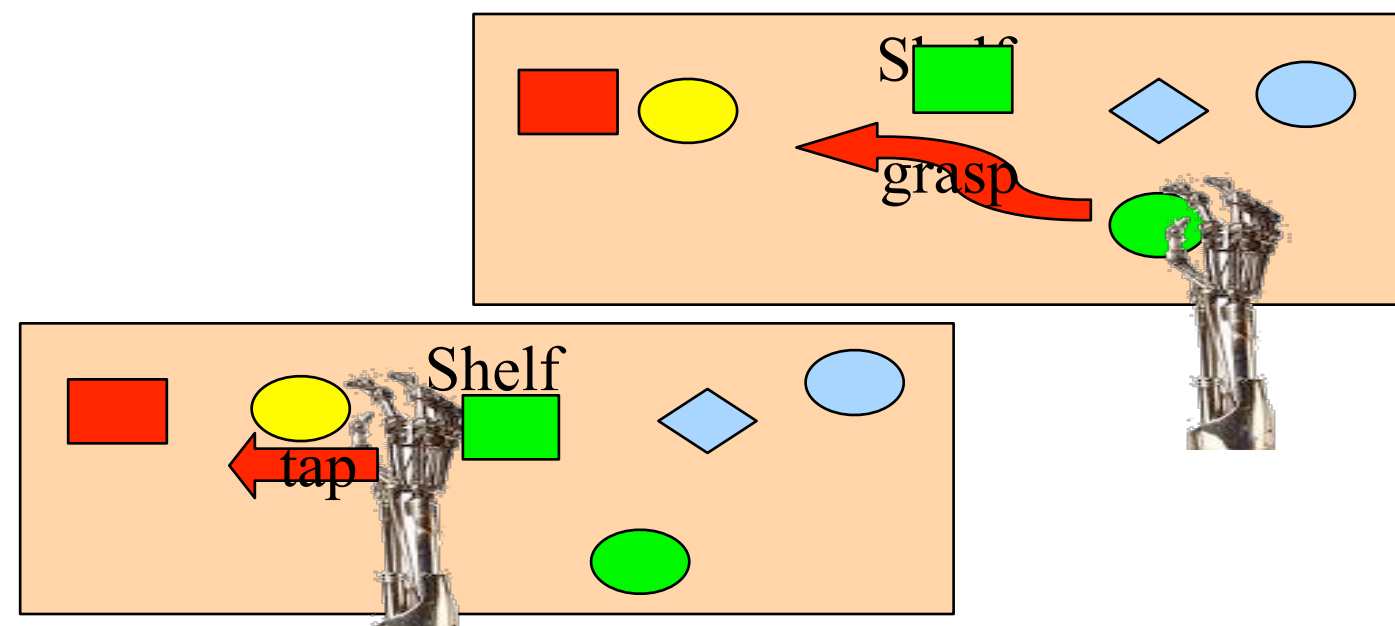
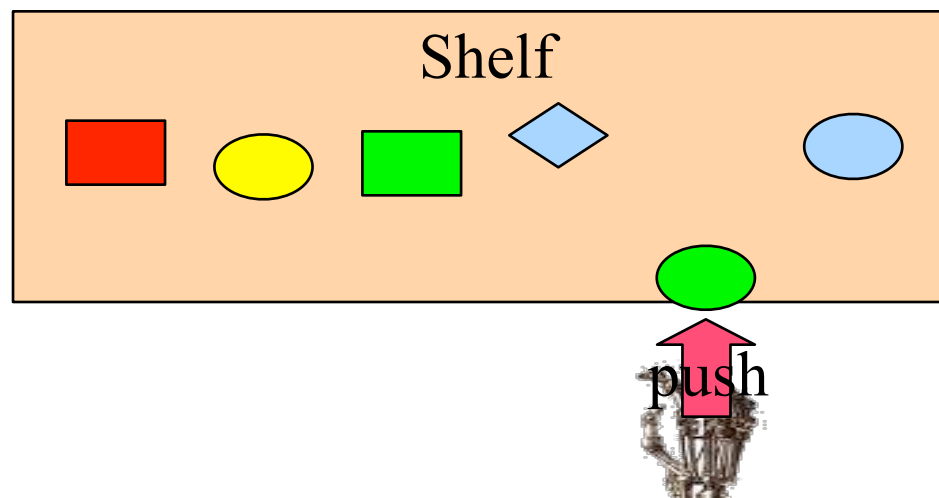


Inputs	Outputs	Function
(O, A)	E	Effect prediction
(O, E)	A	Action recognition/planning
(A, E)	O	Object recognition/selection

Learning relational
affordances
between
two objects
(learnt by experience)

From two object interactions
Generalize to N

Moldovan et al. ICRA 12, 13, 14, PhD 15



**Learning relational
affordances
between
two objects
(learnt by experience)**

Right Arm

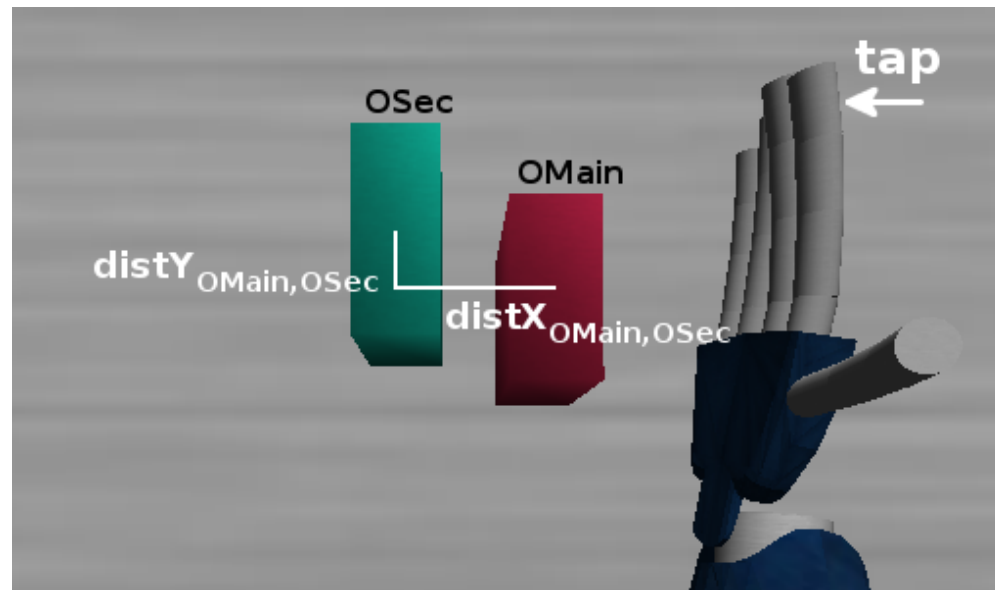
Examples

**Learning relational
affordances
between
two objects
(learnt by experience)**

Right Arm

Examples

What is an affordance ?



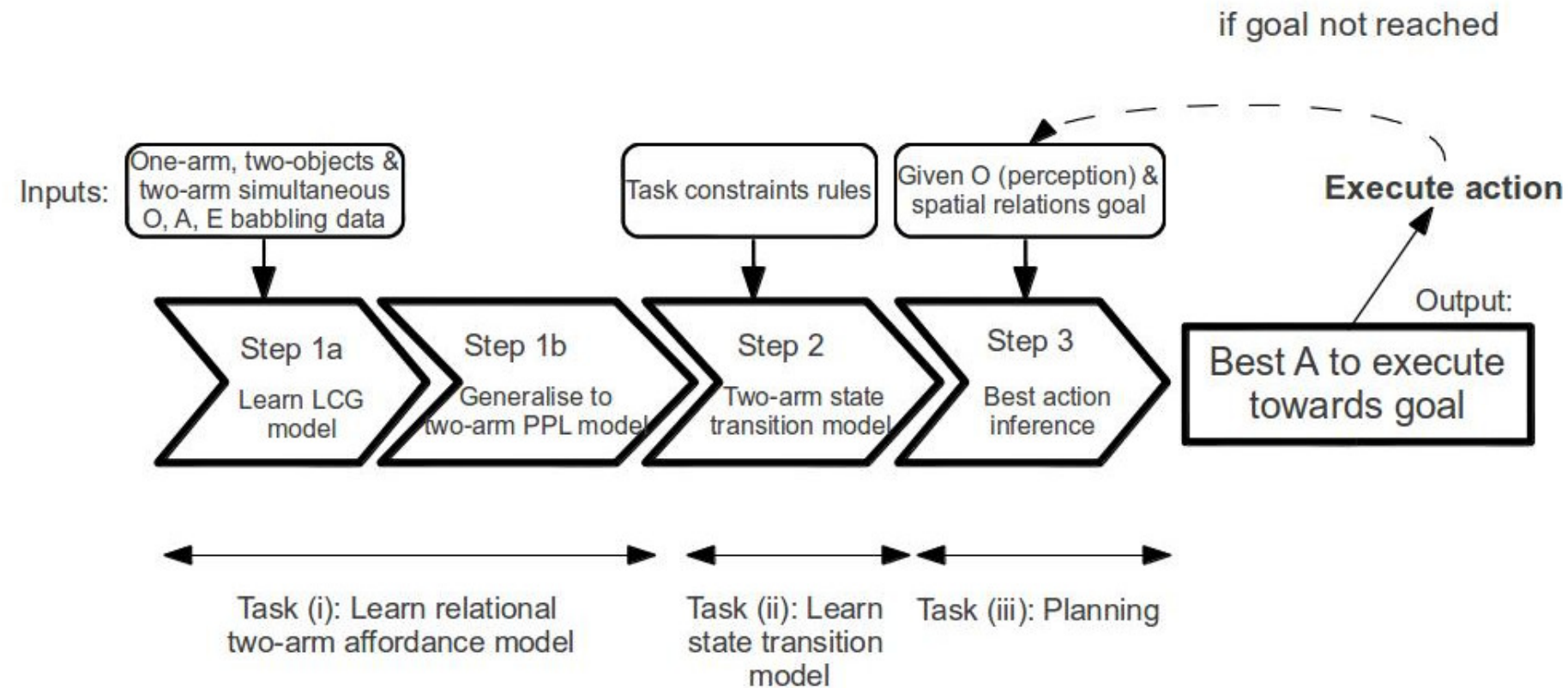
Clip 8: Relational O before (l), and E after the action execution (r).

Table 1: Example collected O , A , E data for action in Figure 8

Object Properties	Action	Effects
$shape_{O_{Main}} : sprism$ $shape_{O_{Sec}} : sprism$ $distX_{O_{Main},O_{Sec}} : 6.94cm$ $distY_{O_{Main},O_{Sec}} : 1.90cm$	$tap(10)$	$displX_{O_{Main}} : 10.33cm$ $displY_{O_{Main}} : -0.68cm$ $displX_{O_{Sec}} : 7.43cm$ $displY_{O_{Sec}} : -1.31cm$

- Formalism — related to STRIPS but models delta
- but also joint probability model over A , E , O

Learning relational affordances

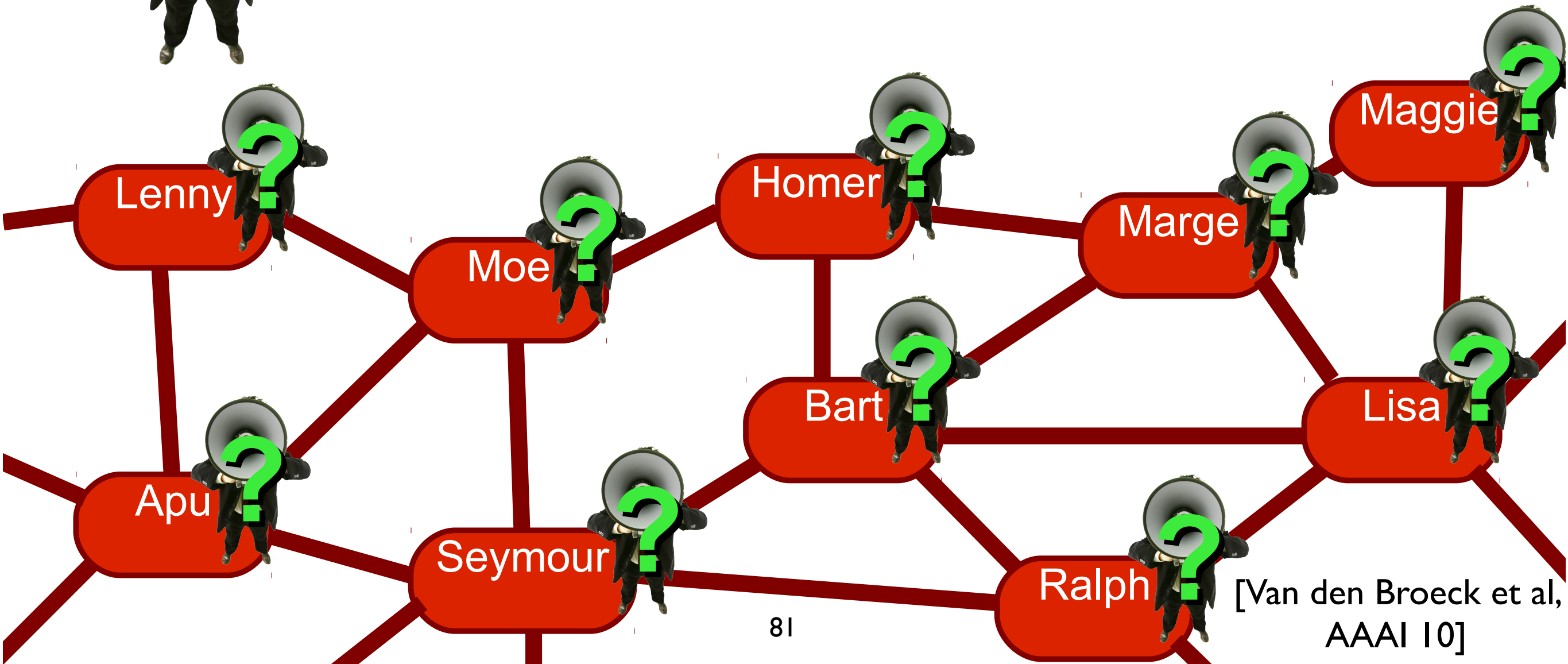
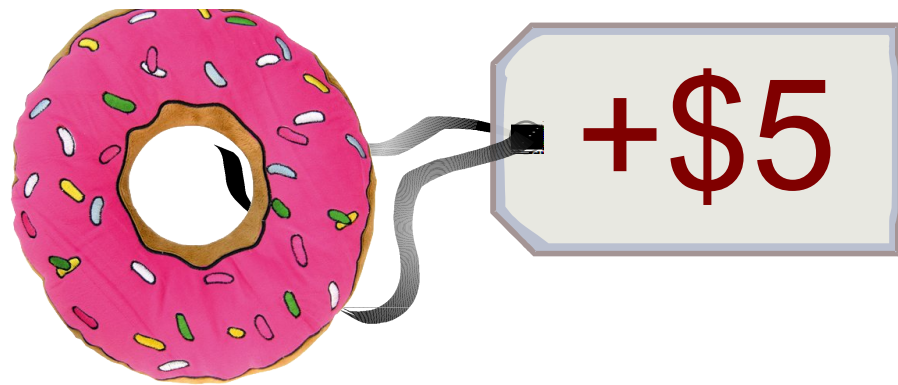


Clip 4: Pipeline for table-top two-arm object manipulation.

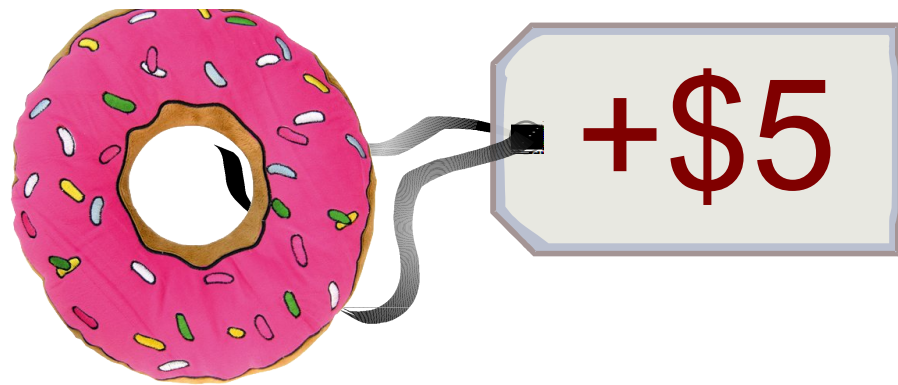
- 1a) learn a Linear Continuous Gaussian (LCG) Bayesian Network (BN) from single arm and simultaneous two-arm exploratory data,
- 1b) from the LCG model, build the two-arm continuous domain relational affordance model in a PPL,
- 2) build a state transition model from the relational affordance model, and
- 3) infer best action to execute to reach goal (step repeated until goal reached).

Viral Marketing

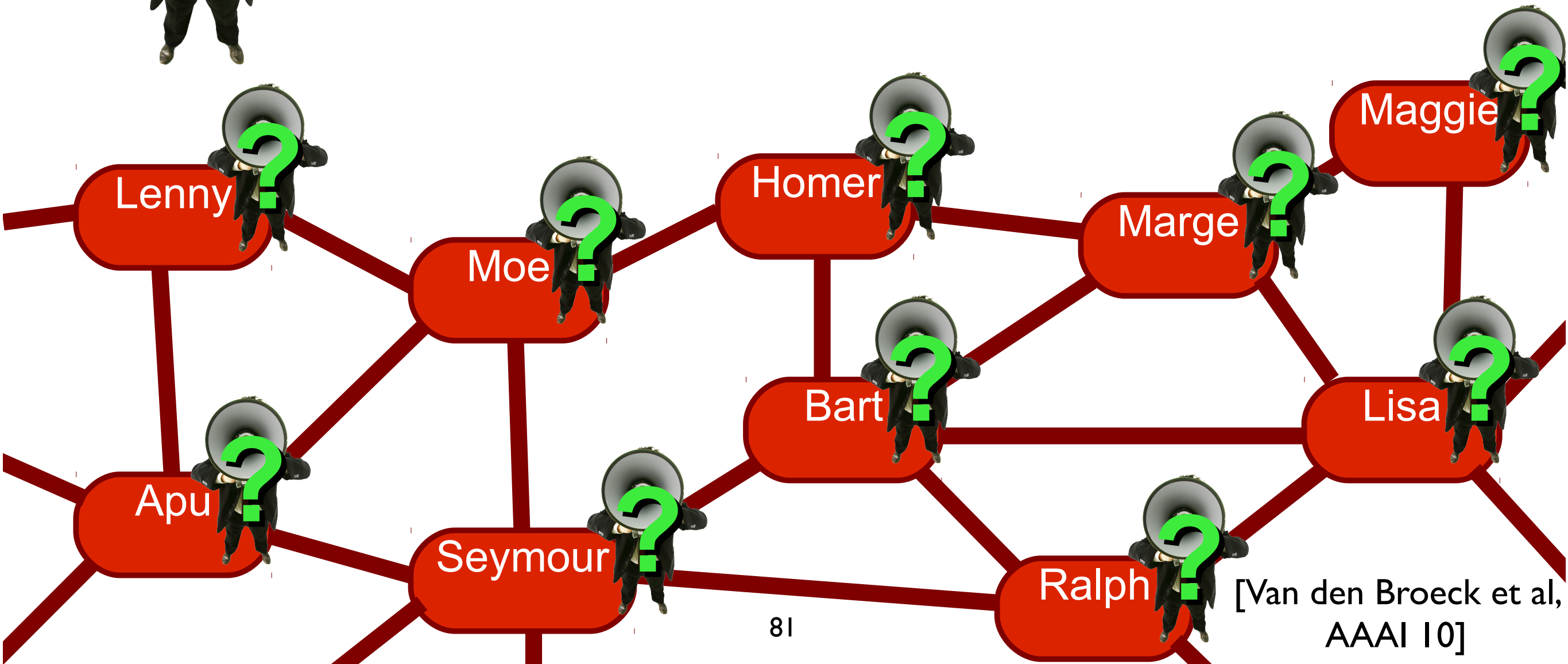
Which advertising strategy maximizes expected profit?



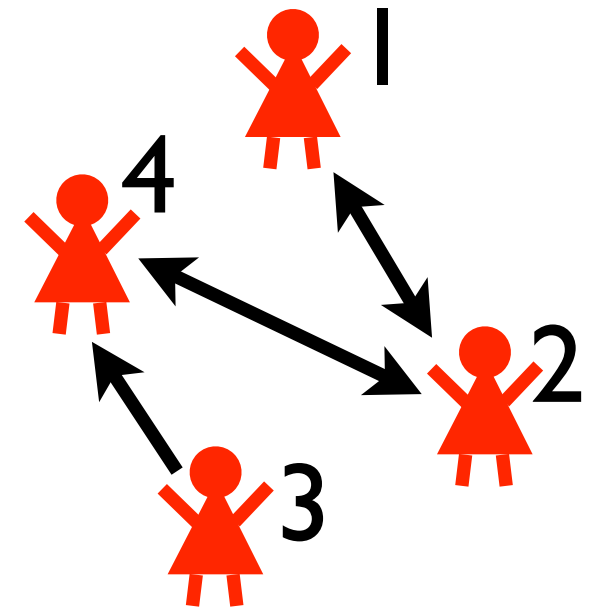
Viral Marketing



decide truth values of
some atoms



DTPProbLog



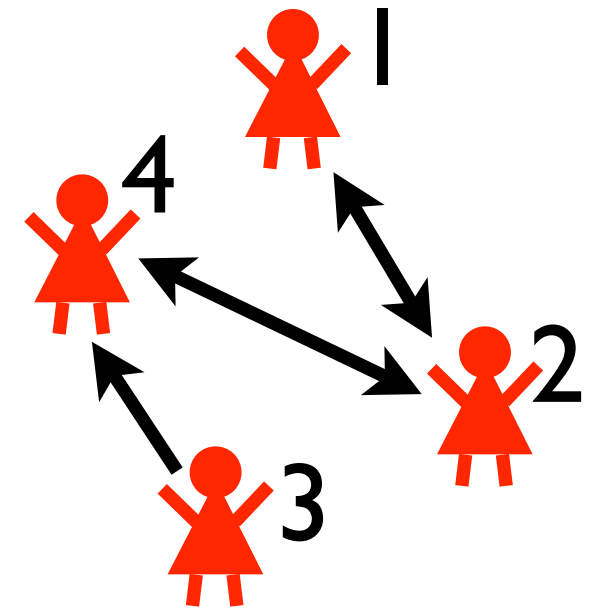
```
person(1).  
person(2).  
person(3).  
person(4).
```

```
friend(1,2).  
friend(2,1).  
friend(2,4).  
friend(3,4).  
friend(4,2).
```

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

decision fact: true or false?



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

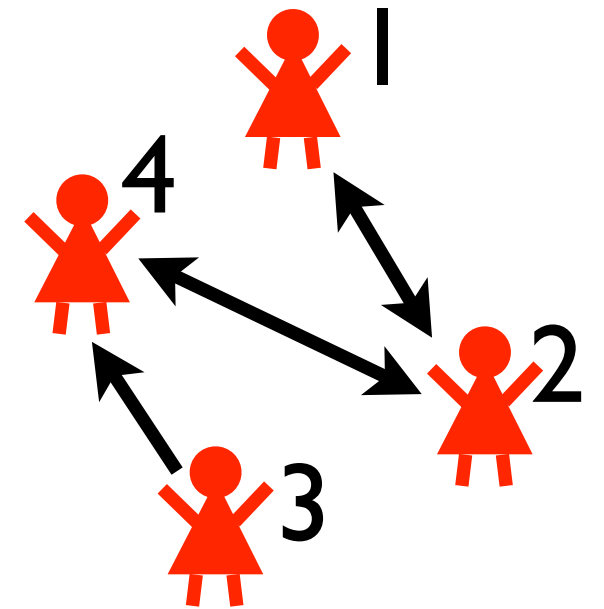
```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

**probabilistic facts
+ logical rules**



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

```
0.2 :: buy_marketing(P) :- person(P) .
```

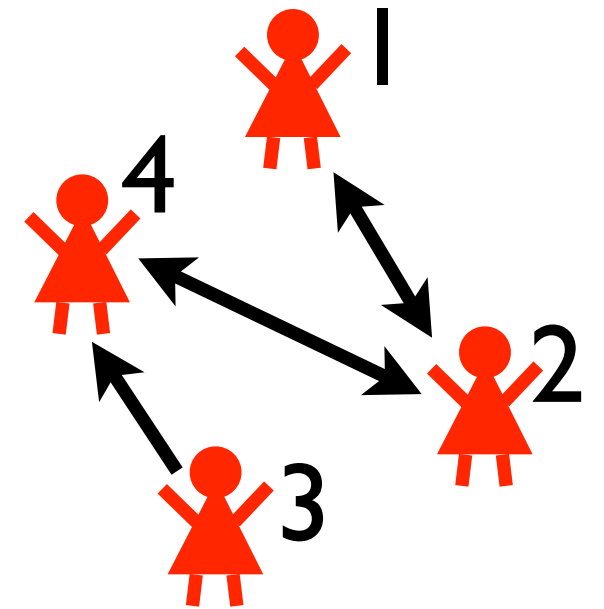
```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```

utility facts: cost/reward if true



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

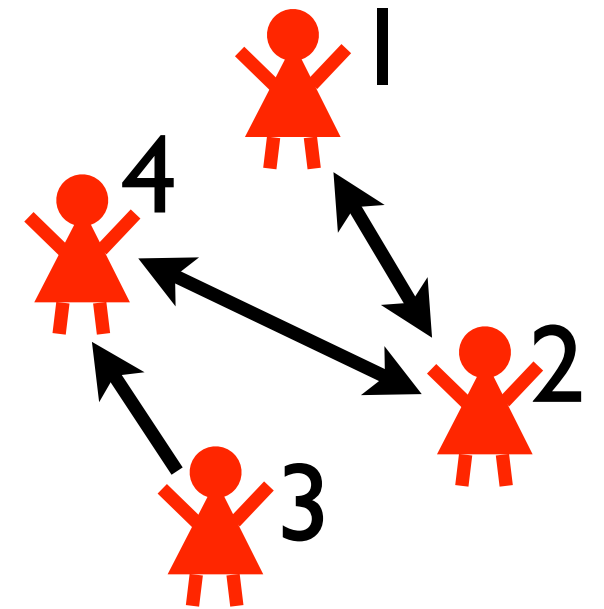
```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

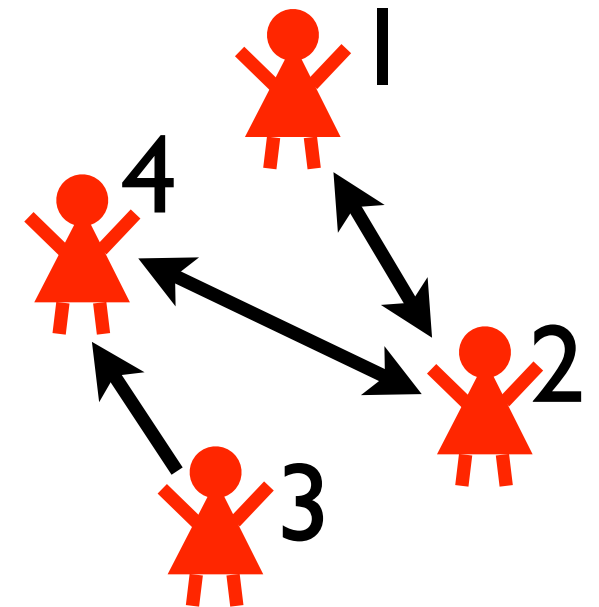
```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

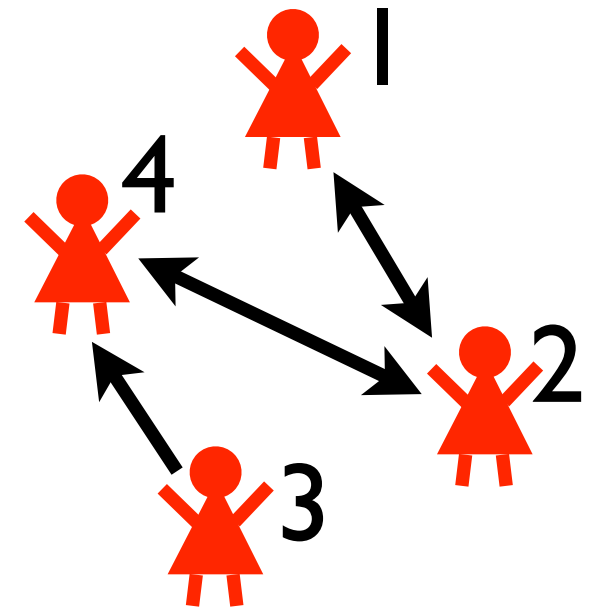
```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

marketed(1)

marketed(3)

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

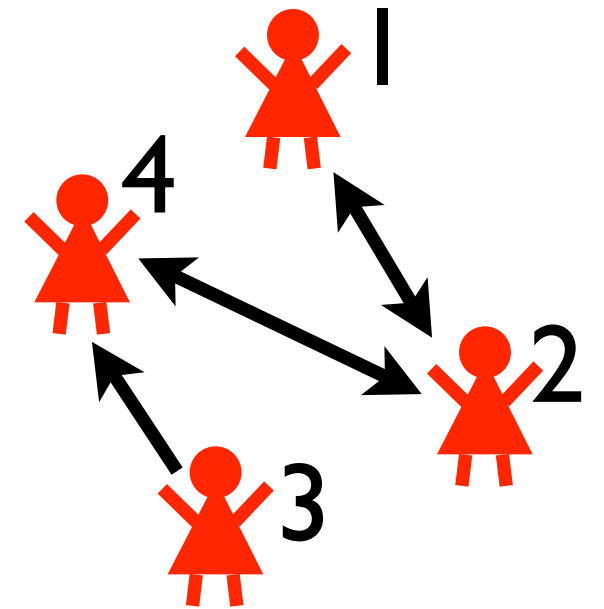
```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

```
marketed(1)
```

```
marketed(3)
```

```
bt(2,1)
```

```
bt(2,4)
```

```
bm(1)
```


DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

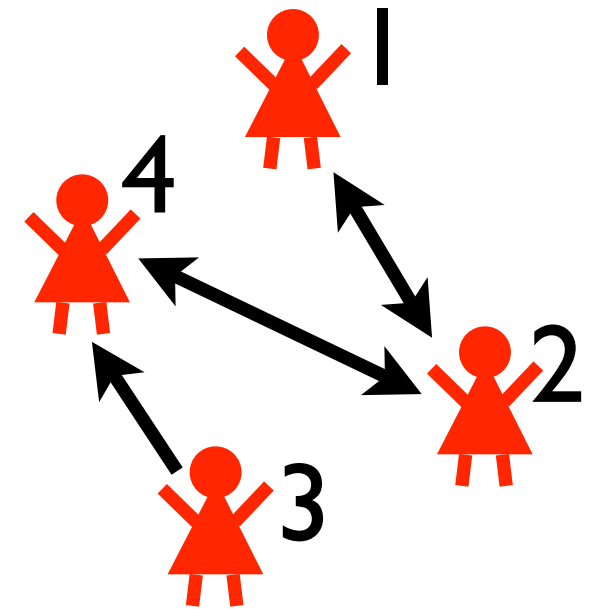
```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

marketed(1)	marketed(3)
bt(2,1)	bt(2,4) bm(1)
buys(1)	buys(2)

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

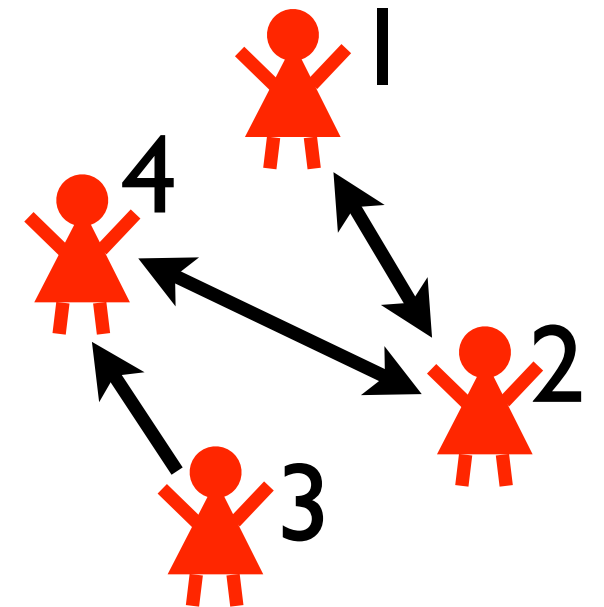
```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```

$$\text{utility} = -3 + -3 + 5 + 5 = 4$$

$$\text{probability} = 0.0032$$

marketed(1)	marketed(3)	
bt(2,1)	bt(2,4)	bm(1)
buys(1)	buys(2)	



```
person(1) .
person(2) .
person(3) .
person(4) .
```

```
friend(1,2) .
friend(2,1) .
friend(2,4) .
friend(3,4) .
friend(4,2) .
```

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```

$$\text{utility} = -3 + -3 + 5 + 5 = 4$$

$$\text{probability} = 0.0032$$

marketed(1)

marketed(3)

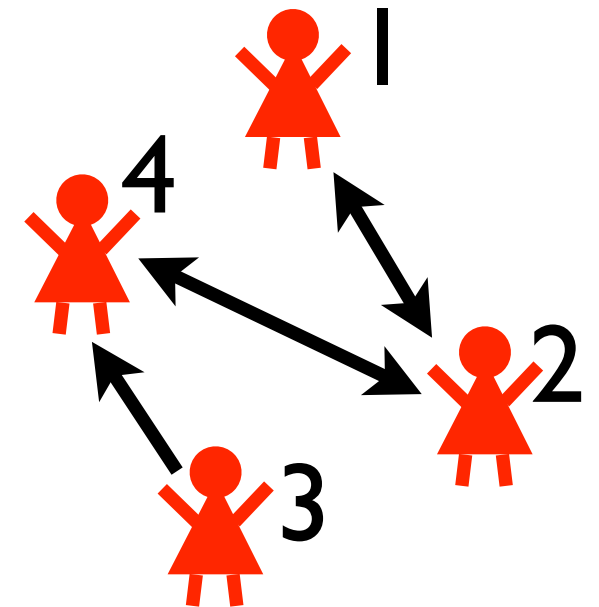
bt(2,1)

bt(2,4)

bm(1)

buys(1)

buys(2)



```
person(1) .
person(2) .
person(3) .
person(4) .
```

```
friend(1,2) .
friend(2,1) .
friend(2,4) .
friend(3,4) .
friend(4,2) .
```

world contributes
 0.0032×4 to
 expected utility of
 strategy

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

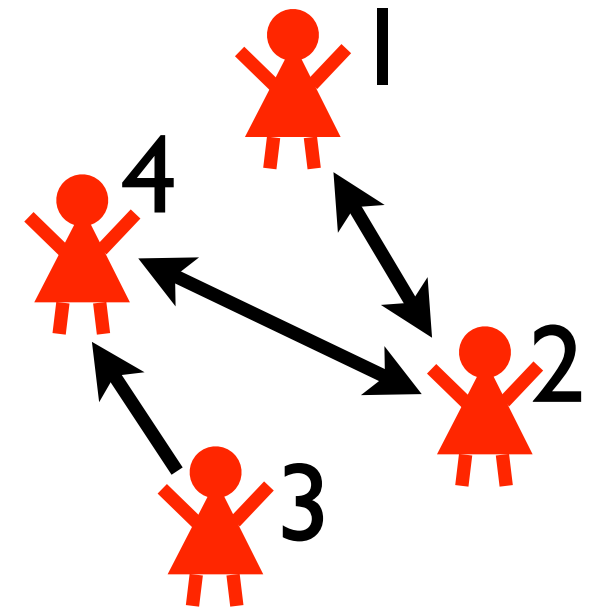
```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```

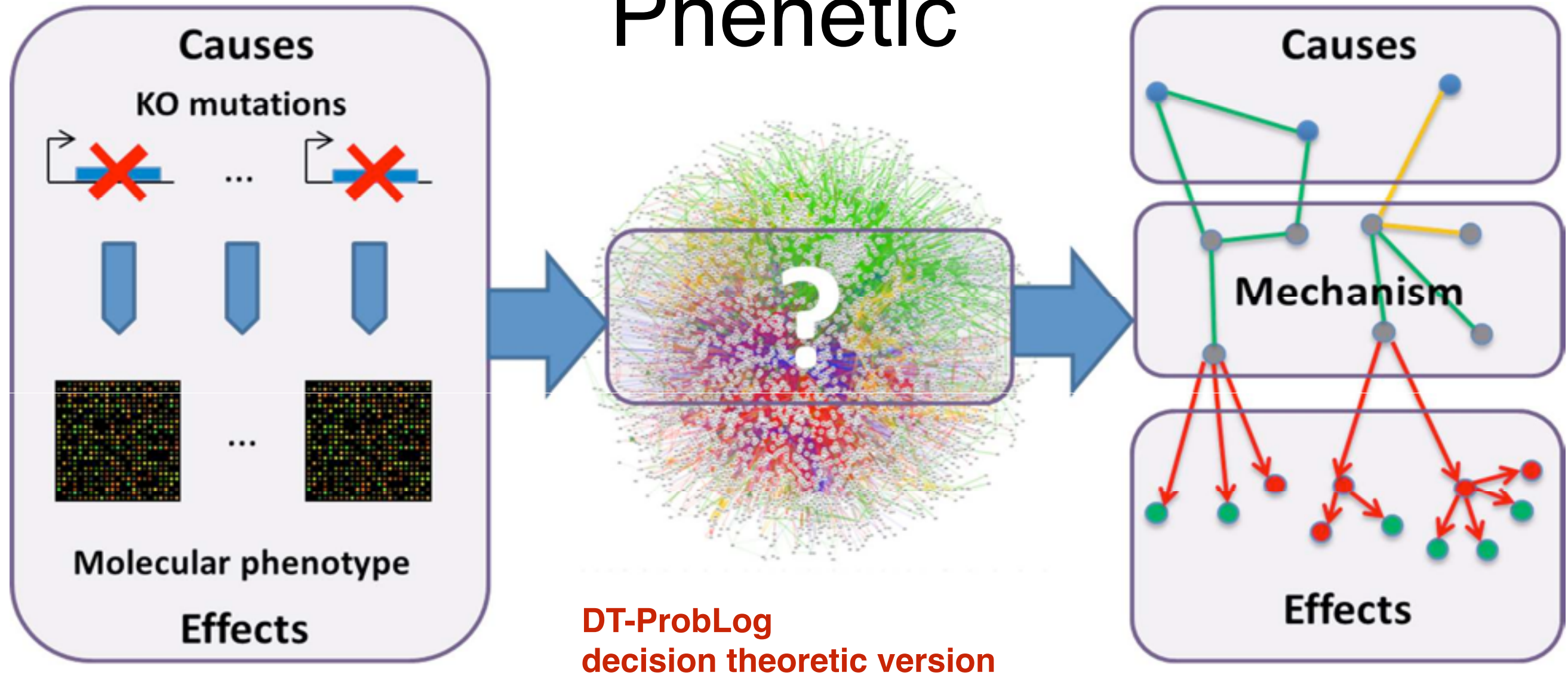


```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

task: find strategy that maximizes expected utility
solution: using ProbLog technology

Phenetic



- Causes: Mutations
 - All related to similar phenotype
- Effects: Differentially expressed genes
 - 27 000 cause effect pairs

- Interaction network:
 - 3063 nodes
 - Genes
 - Proteins
 - 16794 edges
 - Molecular interactions
 - Uncertain

- Goal: connect causes to effects through common subnetwork
 - = Find mechanism
- Techniques:
 - DTProbLog [Van den Broeck]
 - Approximate inference

Can we find the mechanism connecting causes to effects?

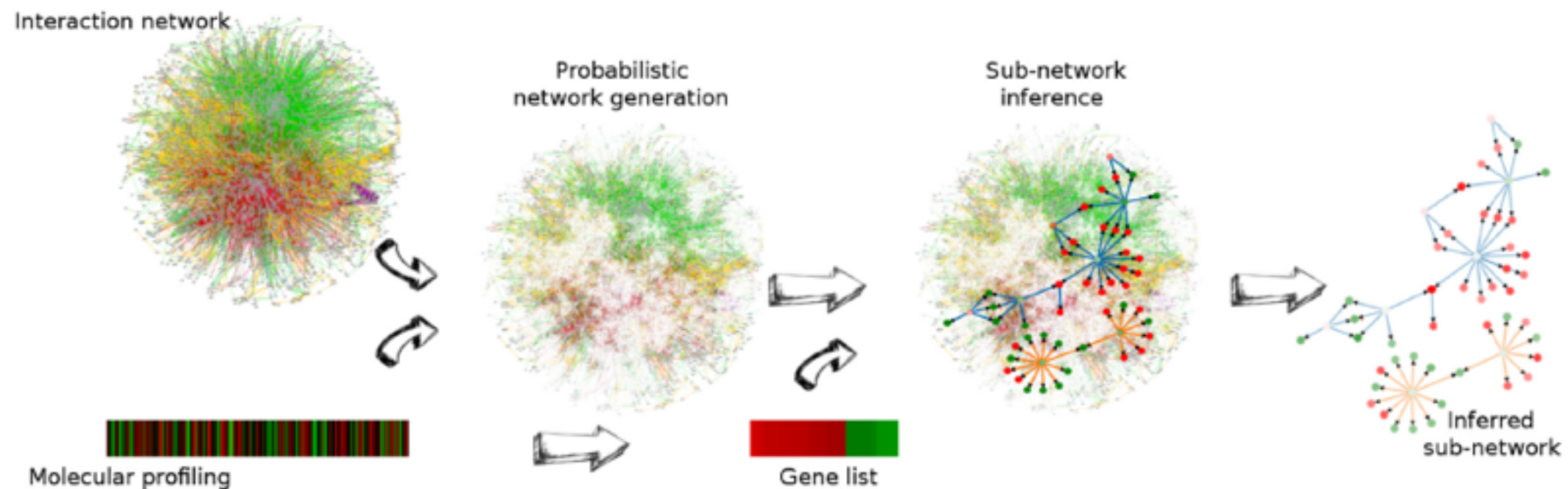


Figure 1. Overview of PheNetic, a web service for network-based interpretation of 'omics' data. The web service uses as input a genome wide interaction network for the organism of interest, a user generated molecular profiling data set and a gene list derived from these data. Interaction networks for a wide variety of organisms are readily available from the web server. Using the uploaded user-generated molecular data the interaction network is converted into a probabilistic network: edges receive a probability proportional to the levels measured for the terminal nodes in the molecular profiling data set. This probabilistic interaction network is used to infer the sub-network that best links the genes from the gene list. The inferred sub-network provides a trade-off between linking as many genes as possible from the gene list and selecting the least number of edges.

[De Mayer et al., NAR 15]

A true application

A tool for Computational Biology

Based on decision theoretic variation of ProbLog

ProbLog / Prob. Programming for prototyping

More specialised inference engine was needed

also some special purpose approximations

A key question in AI:



Statistical relational learning
Probabilistic programming, ...

A key question in AI:

Dealing with uncertainty

- probability theory

models

- Many languages, systems, applications, ...
- not yet a technology ! but a lot of progress
- and a lot more to do !
- ... excellent area for PhDs ...

Statistical relational learning
Probabilistic programming, ...

Maurice Bruynooghe
Bart Demoen
Anton Dries
Daan Fierens
Jason Filippou
Bernd Gutmann
Manfred Jaeger
Gerda Janssens
Kristian Kersting
Angelika Kimmig
Theofrastos Mantadelis
Wannes Meert
Bogdan Moldovan
Siegfried Nijssen
Davide Nitti
Joris Renkens
Kate Revoredo
Ricardo Rocha
Vitor Santos Costa
Dimitar Shterionov
Ingo Thon
Hannu Toivonen
Guy Van den Broeck
Mathias Verbeke
Jonas Vlasselaer

Thanks !

<http://dtai.cs.kuleuven.be/problog>

